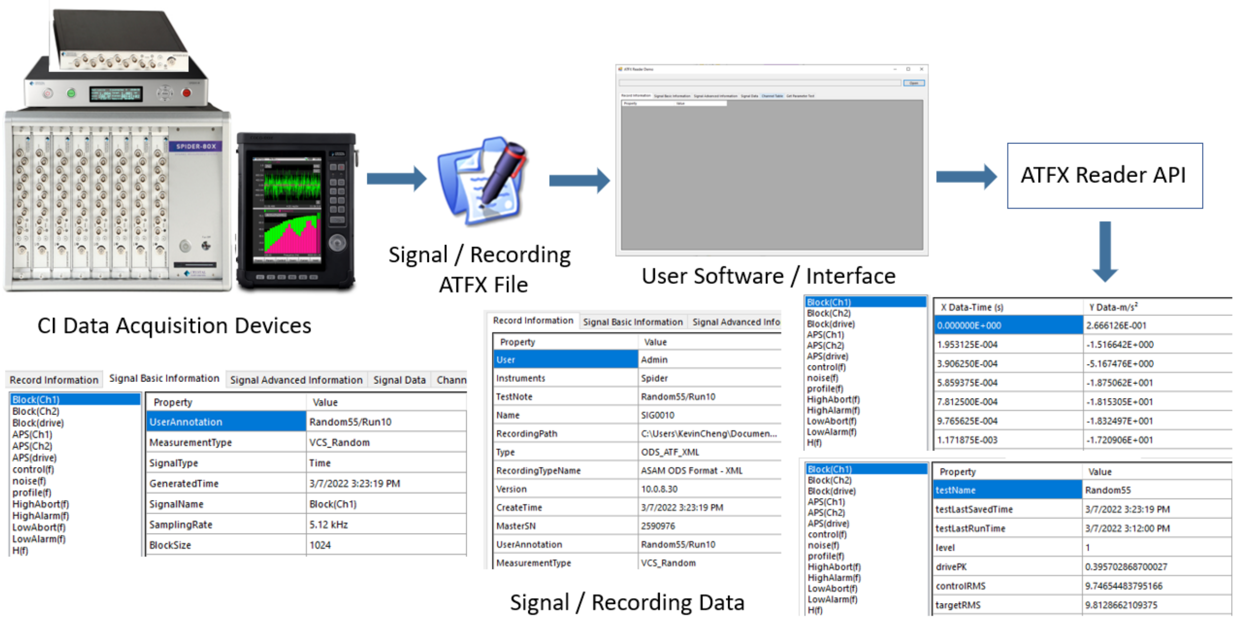
ATFX SIGNAL READER API (C#, PYTHON, MATLAB, LABVIEW)

August 4, 2022

Document ver. 3.6

Contents

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



The Crystal Instruments (CI) ATFX ODS Signal Reader Application Programming Interface (API) consists of two 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.

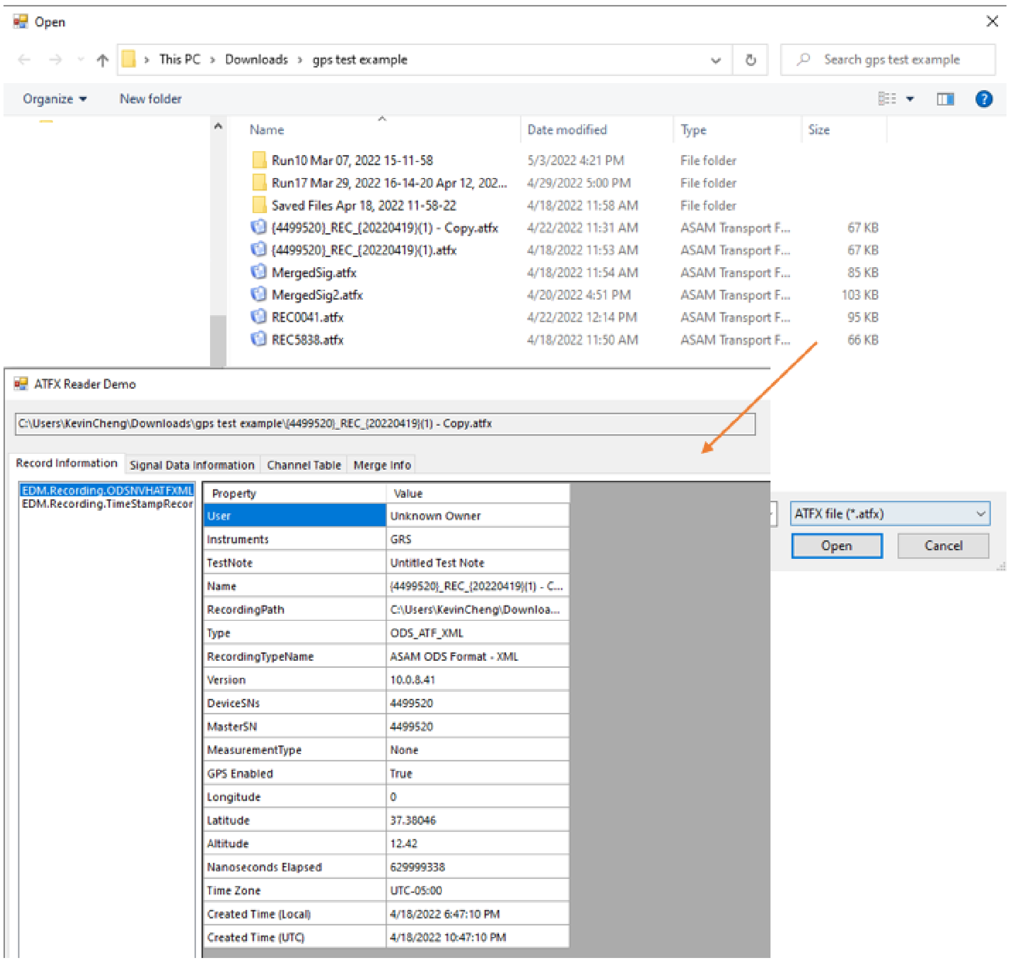
For details about the ATFX ODS format please refer to the official website:

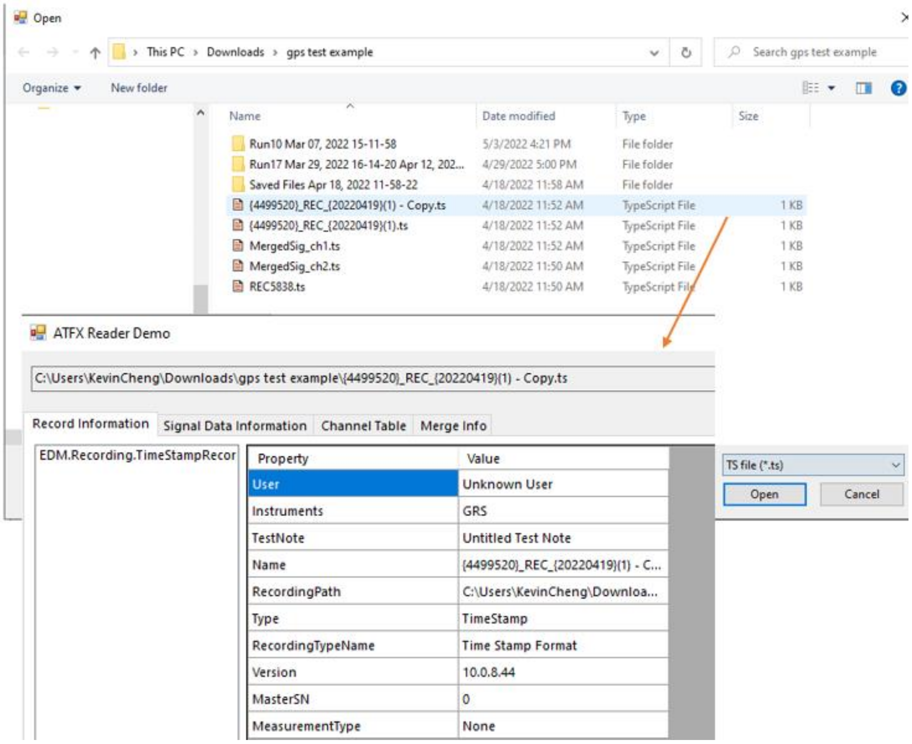
https://www.asam.net/standards/detail/ods/wiki/

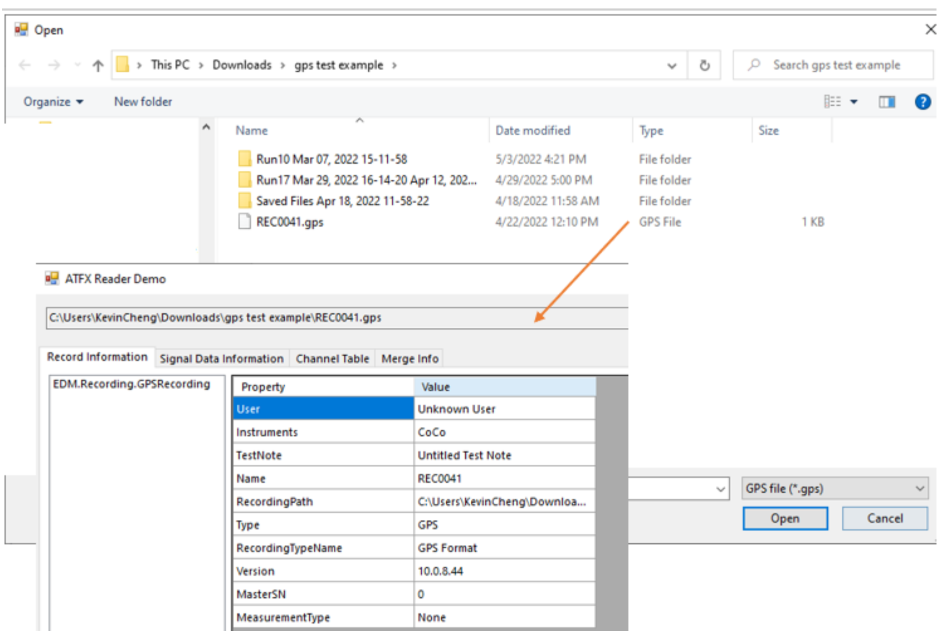
ATFX files are xml-based files which store the signal data along with all the attributes of the signal data including data and time or recording, length of recording, number of channels, channel parameters (e.g., input channel sensor and sensitivities), geographic coordinates, sampling rate, high pass filter, etc.

ATFX files also reference a DAT file that are well-defined for storing both raw time data as well as processed spectral data, calculated from functions including Fourier Transform, Frequency Response Functions, Cross-Power Spectrum, Octave Spectrum, etc. The .dat file is an important part of the ATFX file and, if missing, the ATFX API may not properly read the ATFX file.

There are two additional file types that the .aftx file references that contain 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 application. Currently, we support Windows-based programs, ideally written in C#. The same API also supports Python, MatLab and LabView.

The API offer direct calls to the ASAM ODS model classes and objects used to store data saved in the ATFX file, such as calling the recording NVHMeasurement and NVHEnvironment to read the DateTime with nano seconds elapsed.

The API also provides a Utility class that has methods to return data from the ATFX file without the user needing to understand the complexity of the ASAM ODS model classes. Such as the Utility GetListOfAllSignals that return a list of signals that a ATFX file contains or the Utility GetChannelTable that return a 2D list of strings, where each list is an input channel row.

It is also possible to read any of the signals, time or frequency, in other engineering units (EU), such as Acceleration m/s^2 to g. As well as reading frequency domain signals in other spectrum types, such as EUrms to EUPeak. All done by the signal method GetFrame where users can pass in parameters to return a converted signal frame data saved in the ATFX file.

When the ATFX API read the ATFX file, there may be some differences in the signal frame data, this is due to some display related parameters such as spectrum type not being saved into the ATFX file. By default, the spectrum type is $EUrms^2$. Engineering units are saved into the ATFX file and should be the the default EU when reading the signal frame.

ATFX API Package

Package Contents

Crystal Instruments will provide a **zip file** or **software installer exe 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

Run the installer and it should install the files to the default location:

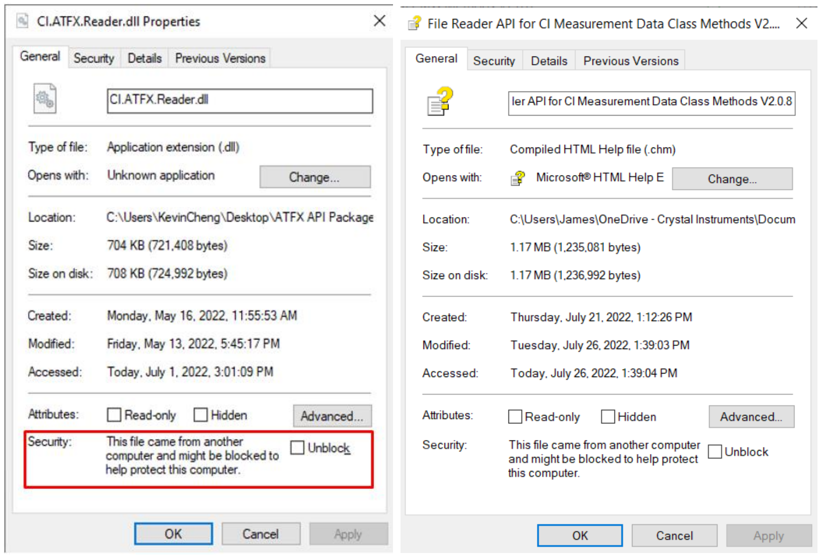
C:\Program Files\Crystal Instruments\Signal Reader API

It is recommended to move any of the coding files outside of the Program Files folder to avoid admin permissions when editing and saving. The dll files can be moved anywhere, so long any custom scripts know the exact file path location of those dll files.

Unreadable DLL Files Despite Correct File Path
Blank CHM File Display Issue

There may be chances where the CHM file displays a blank screen on the right side of the window or a script reading the correct file path and that the dll files exist but throws an error stating that it can not find the dll files. One of the solutions is that in the dll file properties have an additional clickable box or button called **Unblock** and text saying, "This file came from another computer and might be blocked to help protect this computer.". Unblocking the dll file should let the scripts relying on the dll files to be able to find and read them.

This issue occurs because of the computer protecting itself from any files that came from another computer, thus it will sometimes mark files as potentially unsafe and block it so it is not readable.



The C# Demo exe file should fine on its own as it has embedded the dll files into the exe file.

Recommended Versions for Python, Matlab & LabVIEW

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. And a compatible version of Python for the Python.NET package, such as **3.8** or **2.7**. Anything above 3.8 can work by installing a pre-release version of Python.NET.

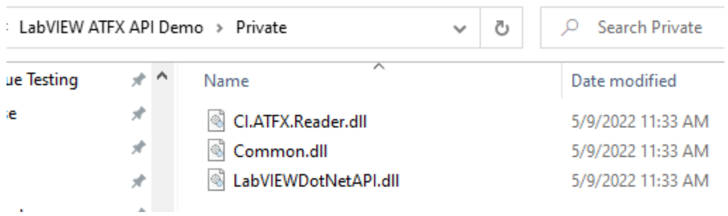
The Python scripts also comes with **Matplotlib** for plotting signal frame data and **Numpy** for converting C# array to Python array.

```
#---Pythonnet clr import
import clr
parentPath = "C:\\Users\\KevinCheng\\ATFX API Package v1.2\\"

clr.AddReference(parentPath + "CI.ATFX.Reader.dll")
clr.AddReference(parentPath + "Common.dll")
```

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

```
% Copyright (C) 2022 by Crystal Instruments Corporation. All rights reserved.
% Load common and reader dll
NET.addAssembly('C:\Users\KevinCheng\ATFX API Package v1.2\Common.dll');
NET.addAssembly('C:\Users\KevinCheng\ATFX API Package v1.2\CI.ATFX.Reader.dll');

%create a atfx recording instance
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 32-bit version**. And use the provided dll files in the LabVIEW ATFX API Demo -> Private folder.


LabVIEW ATFX API Demo › Private
Search Private
ue Testing
Name
Date modified
CI.ATFX.Reader.dll
5/9/2022 11:33 AM
Common.dll
5/9/2022 11:33 AM
LabVIEWDotNetAPI.dll
5/9/2022 11:33 AM

Quick Start

This section of the manual will be focused on a quick reference guide to give the user knowledge of what they need to do. For example, how to read an Auto Power Spectrum signal in C#, Python and Matlab or read the nano seconds from a recording.

Reading a Frequency Domain Signal Frame

Frequency domain data is read from time domain data that is converted through mathematical transforms such as the Fourier Transform.

To read a frequency domain signal, the code must utilize the **ISignal.GetFrame(int index, _SpectrumScalingType spectrumType, string engineeringUnit)** to return a signal frame data. The _SpectrumScalingType and the string format for the engineering units can be found in the **CHM class library file**. Any signal can call the GetFrame method and it will return that signal frame data.

For Real & Imaginary pair spectrum signals, such as Frequency Response Function (FRF), Fast Fourier Transform (FFT) and Cross Power Spectrum (CPS), the Y data may be double the size of the X data. This is because the Real & Imaginary pairs are store together in the Y data, thus the first number of the pair is the Real and the second is the Imaginary.

A frame data example:

Y data frame size: 1024, X data frame size: 512

[0]: Real, [1]: Imaginary, [2]: Real, [3]: Imaginary, … [N]: Real, [N+1]: Imaginary

It is also necessary to call the **ISignal.GetLabel(int dimension)** and **ISignal.GetYLabel()** to get the signal X, Y and Z data labels. The GetYLabel method is the preferred method to get the Y data label for frequency signals, especially for reading Real & Imaginary pairs from FRF, FFT, and CPS. As the GetYLabel will return a list of strings, where the first string is the label for the actual Y data unit and spectrum type, such as $(m/s^2)^2$ (RMS) or Real $(m)/(m/s^2)$. And the second string is the label for the Imaginary of Y data.

Here is a list of frequency signals, their short form, and examples:

Frequency Domain Full Name	EDM / ATFX Abbreviation	Signal Example	
Auto Power Spectrum	APS	APS(Ch#)	HighAbort(f)
		APS(drive)	HighAlarm(f)
		control(f)	LowAbort(f)
		noise(f)	LowAlarm(f)
		profile(f)	
Frequency Response Function	FRF	FRF(Ch#, Ch$)	

	H	H(Ch#, Ch$) H(f) hinv(f)
Fast Fourier Transform	FFT	FFT(Ch#)
Cross Power Spectrum	CPS	CPS(Ch#, Ch$)
Coherence Function	COH	COH(Ch#, Ch$)
Sine Spectrum	Spectrum	Spectrum(Ch#)
Shock Response Spectrum	MaxiSRS PosSRS NegSRS	MaxiSRS(Ch#) PosSRS(Ch#) NegSRS(Ch#)
Order Spectrum	ORDSpec	ORDSpec(Ch#)
Octave Spectrum	OCT	OCT(Ch#)

C# Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Reflection;
using System.Diagnostics;
// DLL file imports
using EDM.RecordingInterface;
using EDM.Recording;
using ASAM.ODS.NVH;
using Common;
using Common.Spider;
using EDM.Utils;

// Set the recording file path and open it to extract a IRecording object
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 ‘APS(Ch4)’
ISignal signalCh4 = signals.Where(sig => sig.Name == 'APS(Ch4)').First();

// Get the signal frame data through the ISignal.GetFrame(int, _SpectrumScalingType,
string)
double[][] frame = signalCh4.GetFrame(0, _SpectrumScalingType.EURMS2,
AccelerationUnitEnumString.ArrayString[AccelerationUnitType.g]);

// Get the X & Y data labels
string xDataLabel = signalCh4.GetLabel(0);
string yDataLabel = signalCh4.GetYLabel()[0];
string zDataLabel;
```

```
// Get the Z data label if it exists
if(frame.Length == 3)
  zDataLabel = signalCh4.GetLabel(2);

// Get the 2nd Y data label is the signal if FRF, FFT, H or CPS
if(signalCh4.Type == SignalType.Frequency && signalCh4.Name != "H(f)" &&
   (signalCh4.Properties.NvhType == _NVHType.FrequencyResponseSpectrum ||
    signalCh4.Properties.NvhType == _NVHType.CrosspowerSpectrum ||
    signalCh4.Properties.NvhType == _NVHType.ComplexSpectrum))
{
  string yDataLabel2 = signalCh4.GetYLabel()[1];
}
```

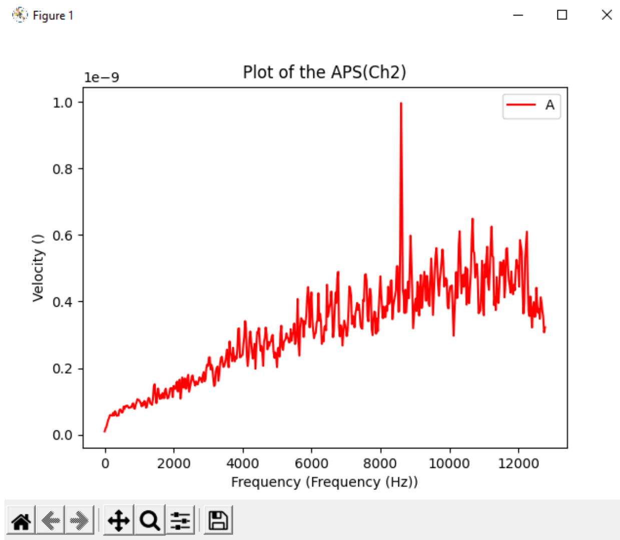
X Data-Frequency (Hz)	Y Data- $(m/s^2)^2$ (RMS)
0	1.22851834021276E-05
25	3.079994712607E-06
50	1.33338728947052E-09
75	1.20776244560972E-09
100	1.25914234594404E-09
125	1.06968833790688E-09
150	1.2482976874395E-09
175	8.62062643491868E-10
200	5.16639009351394E-10
225	3.67680913493373E-10
250	4.44786429909527E-10
275	3.22440490974074E-10

Python Code

```
#---Pythonnet clr import
import clr
# Change file path here to whereever the DLL files are
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')

import numpy as np
import matplotlib.pyplot as plt

#---C# .NET imports & dll imports
from EDM.Recording import *
from EDM.RecordingInterface import *
from ASAM.ODS.NVH import *
from EDM.Utils import *
from Common import *
from Common import _SpectrumScalingType
from Common.Spider import *
from System import *
from System.Diagnostics import *
from System.Reflection import *
from System.Text import *
from System.IO import *
```

```
# Change file path here to whereever signal or recording files are
recordingPath = "C:\\Users\\KevinCheng\\Downloads\\gps test example\\"
# ATFX file path, change contain the file name and correctly reference it in
RecordingManager.Manager.OpenRecording
recordingPathRegular = recordingPath + "SIG0000.atfx"

#OpenRecording(string, out IRecording)
# dummy data is required for the OpenRecording for it to correctly output data
# Make sure to reference the correct file string
dummyTest1, recording = RecordingManager.Manager.OpenRecording(recordingPathRegular,
None)

# Get a list of signals
signalList = Utility.GetListOfAllSignals(recording)

# Get the frame of a frequency signal depending on where it is in the list
# The Convert.ToInt32 is necessary for the the enum AccelerationUnitType to be read as
a int instead of a string
signal = signalList[12]
frame = signal.GetFrame(0, _SpectrumScalingType.EUPeak,
AccelerationUnitEnumString.ArrayString[Convert.ToInt32(AccelerationUnitType.g)])

print("X: ", frame[0][0])
print("Y: ", frame[1][0])
print("X: ", frame[0][1])
print("Y: ", frame[1][1])
print("X: ", frame[0][2])
print("Y: ", frame[1][2])

frameX = np.fromiter(frame[0], float)
frameY = np.fromiter(frame[1], float)

plt.plot(frameX,frameY,'r')
plt.xlabel(signal.Properties.xQuantity + " (" + signal.Properties.xUnit + ")")
plt.ylabel(signal.Properties.yQuantity + " (" + signal.Properties.yUnit + ")")
plt.title("Plot of the " + signal.Name)
plt.legend(signal.Name)
plt.show()
```

```
X:  0.0
Y:  9.586720559615451e-12
X:  25.0
Y:  1.8807570278655703e-11
X:  50.0
Y:  2.335621415745173e-11
```

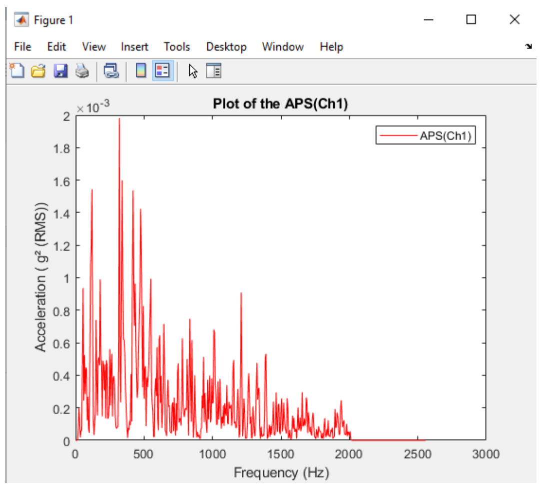


Matlab Code

```
% Load common and reader dll
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');

% Create a atfx recording instance
rec =
EDM.Recording.ODSNVHATFXMLRecording('C:\Users\KevinCheng\Documents\EDM\test\Random69\Run3 Jul 01, 2022 11-20-16\SIG0004.atfx');

% Use item function to get a time signal instance
sig = Item(rec.Signals,9);

% 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));

% Assign the engineering unit
engiUnit =
EDM.RecordingInterface.AccelerationUnitEnumString.ArrayString(System.Convert.ToInt32(EDM.RecordingInterface.AccelerationUnitType.g)+1);
disp(engiUnit);

disp("display signal frame data");
% Get signal frame
frame = sig.GetFrame(0, Common.('_SpectrumScalingType').EURMS2, engiUnit);
% 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));
```

```
Name:APS(Ch1)

X Unit:Hz

Y Unit:m/s²

g

display signal frame data
```



Reading a Time Domain Signal Frame

Time domain data is read from live monitoring of systems and signals in a test over a period of time.

To read a time domain signal, the code must utilize the **ISignal.GetFrame(int index, SpectrumScalingType spectrumType, string engineeringUnit)** to return a signal frame data. While the _SpectrumScalingType is unnecessary for a time domain signal, passing it in the method will not affect the returned frame data. The method offers a parameter to pass in an engineering unit to change the returned frame data. The string format for the engineering units can be found in the **CHM class library file**. Any signal can call the GetFrame method and it will return that signal frame data.

It is also necessary to call the **ISignal.GetLabel(int dimension)** to get the signal X, Y and Z data labels. The ISignal.GetYLabel() can also get the Y data label by referring to the first string in the returned list of strings.

Here is a list of frequency signals, their short form, and examples:

Time Domain Full Name	EDM / ATFX Abbreviation	Signal Example
Time Block	Block	Block(Ch#)
NonEquidistant		Block(drive) control(t) noise(t) profile(t)

C# Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Reflection;
using System.Diagnostics;
// DLL file imports
using EDM.RecordingInterface;
using EDM.Recording;
using ASAM.ODS.NVH;
using Common;
using Common.Spider;
using EDM.Utils;

// Set the recording file path and open it to extract a IRecording object
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 signal frame data through the ISignal.GetFrame(int, _SpectrumScalingType,
string)
double[][] frame = signalCh4.GetFrame(0, _SpectrumScalingType.Unknown,
AccelerationUnitEnumString.ArrayString[AccelerationUnitType.g]);

// Get the X & Y data labels
string xDataLabel = signalCh4.GetLabel(0);
string yDataLabel = signalCh4.GetLabel(1);
string yDataLabelAlt = signalCh4.GetYLabel()[0];
string zDataLabel;

// Get the Z data label if it exists
if(frame.Length == 3)
  zDataLabel = signalCh4.GetLabel(2);
```

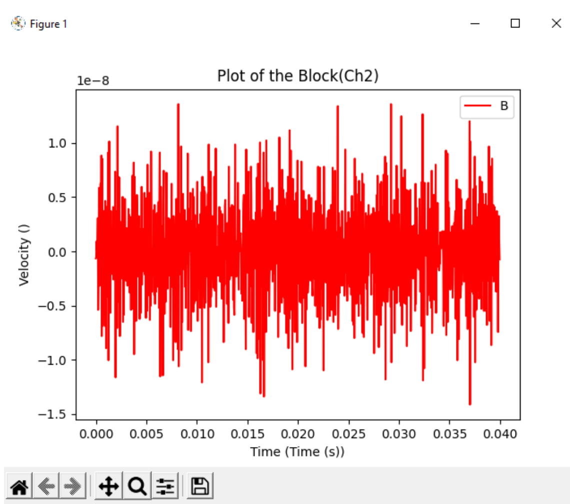
X Data-Time (s)	Y Data-m/s^2
0	-3.83868312835693
0.000195312502910383	-3.18519496917725
0.000390625005820766	2.56844139099121
0.000585937508731149	4.77544021606445
0.000781250011641532	2.94711685180664
0.000976562514551915	2.0478687286377
0.0011718750174623	2.36961460113525
0.00136718752037268	1.12222909927368
0.00156250002328306	-0.055780217051506
0.00175781252619345	2.56172704696655
0.00195312502910383	-0.216037526726723
0.00214843753201421	-3.89411163330078
0.0023437500349246	0.99606454372406
0.00253906253783498	0.984960794448853
0.00273437504074536	-2.72559452056885

Python Code

```
#---Pythonnet clr import
import clr
# Change file path here to whereever the DLL files are
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')

import numpy as np
import matplotlib.pyplot as plt

#---C# .NET imports & dll imports
from EDM.Recording import *
from EDM.RecordingInterface import *
from ASAM.ODS.NVH import *
from EDM.Utils import *
from Common import *
from Common import _SpectrumScalingType
from Common.Spider import *
from System import *
from System.Diagnostics import *
from System.Reflection import *
from System.Text import *
from System.IO import *

# Change file path here to whereever signal or recording files are
recordingPath = "C:\\Users\\KevinCheng\\Downloads\\gps test example\\"
# ATFX file path, change contain the file name and correctly reference it in
RecordingManager.Manager.OpenRecording
recordingPathRegular = recordingPath + "SIG0000.atfx"

#OpenRecording(string, out IRecording)
# dummy data is required for the OpenRecording for it to correctly output data
# Make sure to reference the correct file string
dummyTest1, recording = RecordingManager.Manager.OpenRecording(recordingPathRegular,
None)
```

```
# Get a list of signals
signalList = Utility.GetListOfAllSignals(recording)

# Get the frame of a frequency signal depending on where it is in the list
# The Convert.ToInt32 is necessary for the the enum AccelerationUnitType to be read as a int instead of a string
signal = signalList[4]
frame = signal.GetFrame(0, _SpectrumScalingType.Unknown, AccelerationUnitEnumString.ArrayString[Convert.ToInt32(AccelerationUnitType.g)])

print("X: ", frame[0][0])
print("Y: ", frame[1][0])
print("X: ", frame[0][1])
print("Y: ", frame[1][1])
print("X: ", frame[0][2])
print("Y: ", frame[1][2])

frameX = np.fromiter(frame[0], float)
frameY = np.fromiter(frame[1], float)

plt.plot(frameX,frameY,'r')
plt.xlabel(signal.Properties.xQuantity + " (" + signal.Properties.xUnit + ")")
plt.ylabel(signal.Properties.yQuantity + " (" + signal.Properties.yUnit + ")")
plt.title("Plot of the " + signal.Name)
plt.legend(signal.Name)
plt.show()
```

```
X:  0.0
Y:  -0.00023752757064368662
X:  3.90624991268851e-05
Y:  -0.00023558261631900643
X:  7.81249982537702e-05
Y:  -0.0002474954615576726
```

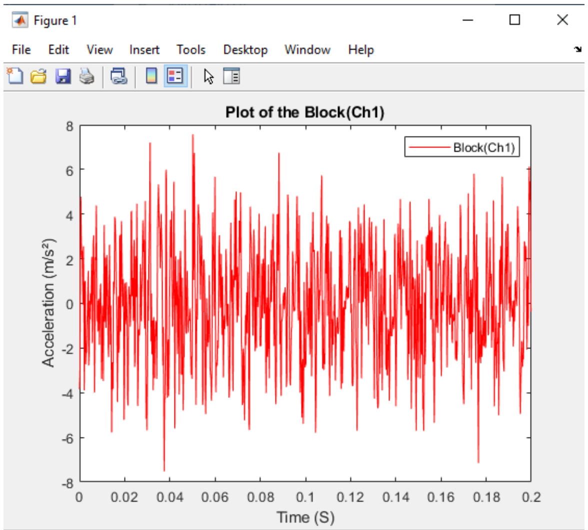


Matlab Code

```
% Load common and reader dll
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');
```

```
% Create a atfx recording instance
rec =
EDM.Recording.ODSNVHATFXMLRecording('C:\Users\KevinCheng\Documents\EDM\test\Random6
9\Run3 Jul 01, 2022 11-20-16\SIG0004.atfx');

% Use item function to get a time 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("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));
```

```
Name:Block(Ch1)

X Unit:S

Y Unit:m/s²

display signal frame data
```



Extracting the Date and Time of a Recording

A recording stores Time and Date in a header file that indicates when the recording was created and saved. For the ATFX file, it stores this information in a DateTime object with accuracy up to millisecond. Sometimes this accuracy is not enough, thus a new data object is created with the purpose of storing better accuracy up to nanoseconds known as DateTimeNano. The DateTimeNano object has a property that stores the millisecond, microsecond and nanosecond together that can be retrieved and separated into each time unit. A .ts file stores the DateTimeNano object that the ATFX file references.

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 referenced in the link below:

https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=net-6.0#fields

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

C# Code

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

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Reflection;
using System.Diagnostics;
// DLL file imports
using EDM.RecordingInterface;
using EDM.Recording;
using ASAM.ODS.NVH;
using Common;
using Common.Spider;
using EDM.Utils;

// Set the recording file path and open it to extract a IRecording object
var recordingPath = “C:\Sig001.atfx”;
RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec);

if (rec is ODSNVHATFXMLRecording nvhRec)
{
  NVHMeasurement nvhMeasurement = nvhRec.Measurement as NVHMeasurement;

  DateTimeNano createTimeLocal = new DateTimeNano(nvhRec.RecordingProperty.CreateTime,
nvhMeasurement.NanoSecondElapsed);
  DateTimeNano createTimeUTC = new
DateTimeNano(Utils.GetUTCTime(nvhRec.RecordingProperty.CreateTime, null),
nvhMeasurement.NanoSecondElapsed);

  bool isNanoTime = createTimeUTC.IsNanoTime;
  uint milli_micro_nano = createTimeUTC.ms_us_ns;
  ulong totalNanoSeconds = createTimeUTC.TotalNanoSeconds;
  string nanoString = 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);
  // Custom Format: yyyy/mm/dd/hh/mm/ss/ms/us/ns
  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);
}
```

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

Python Code

```
#---Pythonnet clr import
import clr
# Change file path here to whereever the DLL files are
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')

#---C# .NET imports & dll imports
from EDM.Recording import *
from EDM.RecordingInterface import *
from ASAM.ODS.NVH import *
from EDM.Utils import *
from Common import *
from Common import _SpectrumScalingType
from Common.Spider import *
from System import *
from System.Diagnostics import *
from System.Reflection import *
from System.Text import *
from System.IO import *

# Change file path here to whereever signal or recording files are
recordingPath = "C:\\Users\\KevinCheng\\Downloads\\gps test example\\"
# ATFX file path, change contain the file name and correctly reference it in
RecordingManager.Manager.OpenRecording
recordingPathRegular = recordingPath + "SIG0000.atfx"

#OpenRecording(string, out IRecording)
# dummy data is required for the OpenRecording for it to correctly output data
# Make sure to reference the correct file string
dummyTest1, recording = RecordingManager.Manager.OpenRecording(recordingPathRegular,
None)

# Create ODS NVH ATFXML Recording object that contains NVH Measurement & NVH
Environment using the file path
```

```
recording = ODSNVHATFXMLRecording(recordingPathRegular)

# If the above created object is ODSNVHATFXMLRecording, it should be able to get the
NVH Measurement & NVH Environment and assigned them
if type(recording) is ODSNVHATFXMLRecording:
    nvhRec = recording
    nvhMeasurement = nvhRec.Measurement

    # Create DateTimeNano objects for local and UTC time zones
    createTimeLocal = DateTimeNano(nvhRec.RecordingProperty.CreateTime,
nvhMeasurement.NanoSecondElapsed)
    createTimeUTC = DateTimeNano(Utils.GetUTCTime(nvhRec.RecordingProperty.CreateTime,
None), nvhMeasurement.NanoSecondElapsed)

    print("Printing UTC")
    print(createTimeUTC.IsNanoTime)
    print(createTimeUTC.ms_us_ns)
    print(createTimeUTC.TotalNanoSeconds)
    print(createTimeUTC.ToNanoString())

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

    print("\nPrinting local")
    print(createTimeLocal.IsNanoTime)
    print(createTimeLocal.ms_us_ns)
    print(createTimeLocal.TotalNanoSeconds)
    print(createTimeLocal.ToNanoString())

    ms = createTimeUTC.ms_us_ns / 1e6
    us = createTimeUTC.ms_us_ns / 1e3 % 1e3
    ns = createTimeUTC.ms_us_ns % 1e3
    # Custom Format: yyyy/mm/dd/hh/mm/ss/ms/us/ns
    print("{0}/{1}/{2}/{3}/{4}/{5}/{6}/{7}/{8}".format(createTimeLocal.Year,
createTimeLocal.Month, createTimeLocal.Day, createTimeLocal.Hour,
createTimeLocal.Minute, createTimeLocal.Second, ms, us, ns))
```

```
Printing UTC
True
629999338
67630629999338
4/18/2022 6:47:10 PM.629.999.338
2022/4/18/18/47/10/629.999338/999.3379999999888/338.0

Printing local
True
629999338
53230629999338
4/18/2022 2:47:10 PM.629.999.338
2022/4/18/14/47/10/629.999338/999.3379999999888/338.0
```

Matlab Code

```
% Load common and reader dll
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');

% Create a atfx recording instance
rec = EDM.Recording.ODSNVHATFXMLRecording('C:\Users\KevinCheng\Downloads\gps test example\{4499520}_REC_{20220419}(1).atfx');

% Assign the NVH Measurement and NVH Environment
nvhMeasurement = rec.Measurement;

% Create the DateTimeNano in UTC and or Local
createTimeLocal = Common.DateTimeNano(rec.RecordingProperty.CreateTime,
nvhMeasurement.NanoSecondElapsed);
createTimeUTC =
Common.DateTimeNano(Common.Utils.GetUTCTime(rec.RecordingProperty.CreateTime, []),
nvhMeasurement.NanoSecondElapsed);

% Display nano type properties
disp('Printing UTC');
disp(createTimeUTC.IsNanoTime);
disp(createTimeUTC.ms_us_ns);
disp(createTimeUTC.TotalNanoSeconds);
disp(createTimeUTC.ToNanoString());

ms = (createTimeUTC.ms_us_ns - rem(createTimeUTC.ms_us_ns,  1e6)) / 1e6;
us = rem(createTimeUTC.ms_us_ns / 1e3, 1e3);
ns = rem(createTimeUTC.ms_us_ns, 1e3);

% Custom Format: yyyy/mm/dd/hh/mm/ss/ms/us/ns
str = sprintf('%d/%d/%d/%d/%d/%d/%d/%d/%d', createTimeUTC.Year,
createTimeUTC.Month, createTimeUTC.Day, createTimeUTC.Hour, createTimeUTC.Minute,
createTimeUTC.Second, ms, us, ns);
disp(str);

% Display nano type properties
disp('Printing local');
disp(createTimeLocal.IsNanoTime);
disp(createTimeLocal.ms_us_ns);
disp(createTimeLocal.TotalNanoSeconds);
disp(createTimeLocal.ToNanoString());

ms = (createTimeLocal.ms_us_ns - rem(createTimeLocal.ms_us_ns,  1e6)) / 1e6;
us = rem(createTimeLocal.ms_us_ns / 1e3, 1e3);
ns = rem(createTimeLocal.ms_us_ns, 1e3);

% Custom Format: yyyy/mm/dd/hh/mm/ss/ms/us/ns
str = sprintf('%d/%d/%d/%d/%d/%d/%d/%d/%d', createTimeLocal.Year,
createTimeLocal.Month, createTimeLocal.Day, createTimeLocal.Hour,
createTimeLocal.Minute, createTimeLocal.Second, ms, us, ns);
disp(str);
```

```
Printing UTC
   1

   629999338

   67630629999338

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

2022/4/18/18/47/10/629/999/338
Printing local
   1

   629999338

   53230629999338

4/18/2022 2:47:10 PM.629.999.338

2022/4/18/14/47/10/629/999/338
``
```

Reading GPS Data from a ATFX File

A recording recorded in a device that can record GPS data such as the Crystal Instruments Ground Recording System (CI-GRS) can save location data into a .gps file that the ATFX file references.

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 methods:

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

C# Code

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

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Reflection;
using System.Diagnostics;
// DLL file imports
using EDM.RecordingInterface;
using EDM.Recording;
using ASAM.ODS.NVH;
using Common;
using Common.Spider;
using EDM.Utils;

// Set the recording file path and open it to extract a IRecording object
var recordingPath = "C:\Sig001.atfx";
RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec);

if (rec is ODSNVHATFXMLRecording nvhRec)
{
  NVHMeasurement nvhMeasurement = nvhRec.Measurement as NVHMeasurement;
```

```
    NVHEnvironment nvhEnvironment = nvhRec.Environment as NVHEnvironment;

    bool bGPS = nvhMeasurement.GPSEnabled;
    double lng;
    double lat;
    double alt;
    double nano;
    string timeZone;
    string softwareVer;
    string hardwareVer;
    string firmwareVer;
    string bitVer;

    if (bGPS)
    {
      lng = nvhMeasurement.Longitude;
      lat = nvhMeasurement.Latitude;
      alt = nvhMeasurement.Altitude;
      nano = nvhMeasurement.NanoSecondElapsed;
    }

    if (!String.IsNullOrEmpty(nvhEnvironment.TimeZone))
    {
      timeZone = nvhEnvironment.TimeZone;
    }

    DateTime creaTimeLocal = nvhRec.RecordingProperty.CreateTime;
    DateTime creaTimeUTC = Utils.GetUTCTime(nvhRec.RecordingProperty.CreateTime, null);

    if (!String.IsNullOrEmpty(nvhEnvironment.InstruSoftwareVersion))
    {
      softwareVer = nvhEnvironment.InstruSoftwareVersion;
      hardwareVer = nvhEnvironment.HardwareVersion;
      firmwareVer = nvhEnvironment.FirmwareVersion;
      bitVer = nvhEnvironment.BitVersion;
    }
  }
```

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

Python Code

```
#---Pythonnet clr import
import clr
# Change file path here to whereever the DLL files are
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')

#---C# .NET imports & dll imports
from EDM.Recording import *
from EDM.RecordingInterface import *
from ASAM.ODS.NVH import *
from EDM.Utils import *
from Common import *
from Common import _SpectrumScalingType
from Common.Spider import *
from System import *
from System.Diagnostics import *
from System.Reflection import *
from System.Text import *
from System.IO import *

# Change file path here to whereever signal or recording files are
recordingPath = "C:\\Users\\KevinCheng\\Downloads\\gps test example\\"
# ATFX file path, change contain the file name and correctly reference it in
RecordingManager.Manager.OpenRecording
recordingPathRegular = recordingPath + "SIG0000.atfx"

#OpenRecording(string, out IRecording)
# dummy data is required for the OpenRecording for it to correctly output data
# Make sure to reference the correct file string
```

```
dummyTest1, recording = RecordingManager.Manager.OpenRecording(recordingPathRegular,
None)

# Create ODS NVH ATFXML Recording object that contains NVH Measurement & NVH
Environment using the file path
recording = ODSNVHATFXMLRecording(recordingPathRegular)

# If the above created object is ODSNVHATFXMLRecording, it should be able to get the
NVH Measurement & NVH Environment and assigned them
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(nvhEnvironment.TimeZoneString):
        print("Time Zone: ", nvhEnvironment.TimeZoneString)

    print("Created Time (Local): ", nvhRec.RecordingProperty.CreateTime)
    print("Created Time (UTC): ", Utils.GetUTCTime(nvhRec.RecordingProperty.CreateTime,
None))
    dateTimeNano = DateTimeNano(nvhRec.RecordingProperty.CreateTime,
UInt32(nvhMeasurement.NanoSecondElapsed))
    print("DateTimeNano Object: ", dateTimeNano)
```

```
GPS Enabled:  True
Longitude:  0.0
Latitude:  37.38046
Altitude:  12.42
Nanoseconds Elapsed:  629999338
Time Zone:  Eastern Standard Time;-300;(UTC-05:00) Eastern Time (US & Canada);Eastern Standard Time;Eastern Daylight Time;[01:01:0001;12:31:2
006;60;[0;02:00:00;4;1;0;];[0;02:00:00;10;5;0;];][01:01:2007;12:31:9999;60;[0;02:00:00;3;2;0;];[0;02:00:00;11;1;0;];];
Created Time (Local):  4/18/2022 2:47:10 PM
Created Time (UTC):  4/18/2022 6:47:10 PM
DateTimeNano Object:  4/18/2022 2:47:10 PM.629.999.338
```

Matlab Code

```
% Load common and reader dll
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');

% Create a atfx recording instance
rec = EDM.Recording.ODSNVHATFXMLRecording('C:\Users\KevinCheng\Downloads\gps test
example\{4499520}_REC_{20220419}(1).atfx');

% Display gps properties
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.TimeZoneString));
disp(System.String.Format("Created Time
(Local):{0}",rec.RecordingProperty.CreateTime));
disp(System.String.Format("Created Time (UTC):{0}",
Common.Utils.GetUTCTime(rec.RecordingProperty.CreateTime, [])));
disp(System.String.Format("Nanoseconds
Elapsed:{0}",rec.Measurement.NanoSecondElapsed));
```

```
GPS Enable:True

Longitude:0

Latitude:37.38046

Altitude:12.42

Time zone:Eastern Standard Time;-300;(UTC-05:00) Eastern Time (US & Canada);Eastern Standard Time;Ea

Created Time (Local):4/18/2022 2:47:10 PM

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

Nanoseconds Elapsed:629999338
```

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.

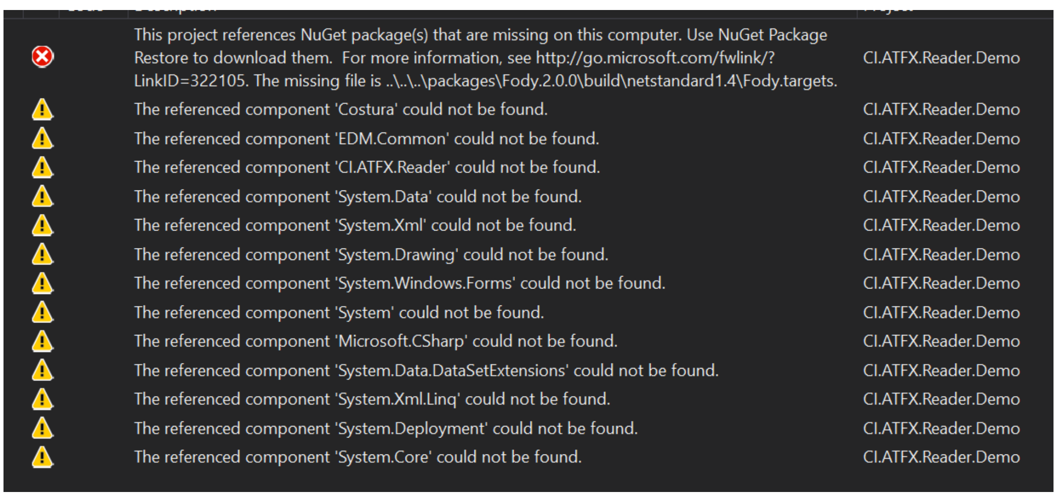
There are 3 file types that the ATFX API can open: .atfx, .ts and .gps. The .atfx is the header file that references .dat, which contains all of the signal frame data and other data not referenced in the .atfx file. It can also reference .ts and .gps files. The .dat file is an important part of the ATFX file and if it is missing the ATFX API may not be able to properly read the ATFX file.

There may be a chance that the data displayed in the ATFX API is different from what is displayed on EDM. This is due to the spectrum type being a display parameter and not saved in the ATFX file, thus it will default to $EUrms^2$.

The demo should load the initial saved engineering units when reading any of the signal frame data.

Building the C# Demo

When opening the C# demo csproj file in Visual Studio, there may be issues that come up such as missing component reference warnings or an error about a missing package file.



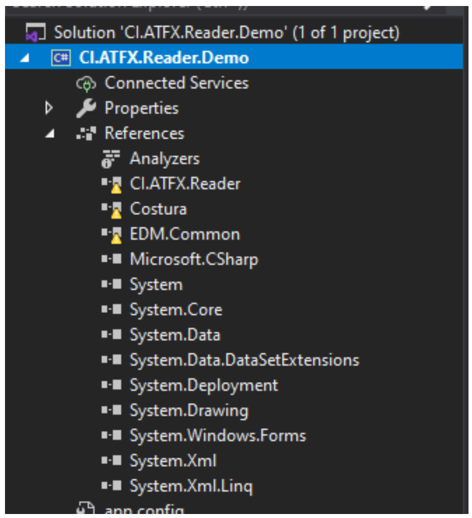
First, open the csproj file in notepad, locate the target block code and remove it. It should be near the bottom of the file.

```
<Target Name="EnsureNuGetPackageBuildImports" BeforeTargets="PrepareForBuild">
  <PropertyGroup>
    <ErrorText>This project references NuGet package(s) that are missing on this computer. Use NuGet Package Restore to download them.  For more information, see http://go.microsoft.com/fwlink/?LinkID=322105. The missing file is {0}.</ErrorText>
```

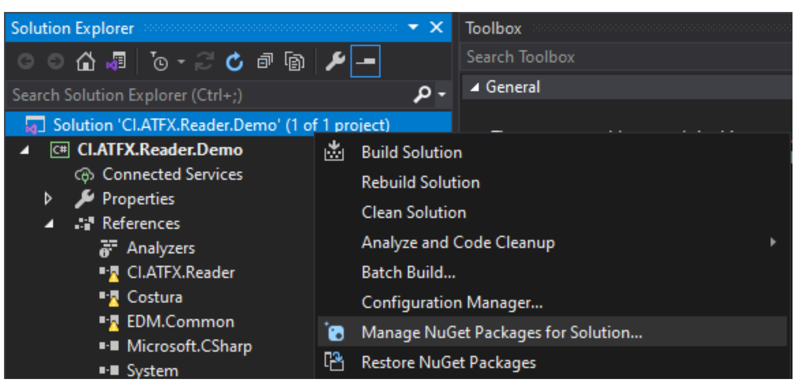
```
  </PropertyGroup>
  <Error Condition="!Exists('..\..\..\packages\Fody.2.0.0\build\netstandard1.4\Fody.targets')" Text="$([System.String]::Format('$(ErrorText)', '..\..\..\packages\Fody.2.0.0\build\netstandard1.4\Fody.targets'))" />
  <Error Condition="!Exists('..\..\..\packages\Costura.Fody.1.6.2\build\dotnet\Costura.Fody.targets')" Text="$([System.String]::Format('$(ErrorText)', '..\..\..\packages\Costura.Fody.1.6.2\build\dotnet\Costura.Fody.targets'))" />
</Target>
```

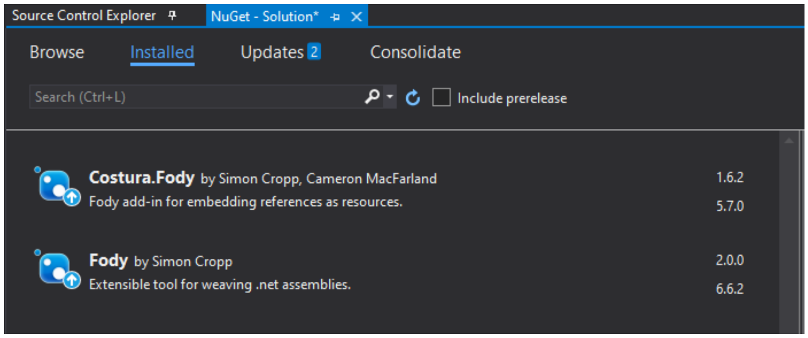
Save the file and reload the visual studio when the prompt comes up.

The system related components should be fixed:



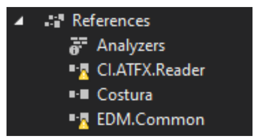
Save the solution file where the csproj file is located then right click the solution or project file in Visual Studio Solution Explorer -> Manage Nuget Packages.



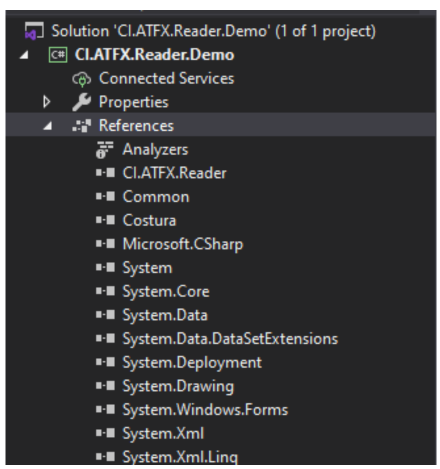


Uninstall the Costura.Fody v1.6.2 and Fody v2.0.0 packages and reinstall in them to fix the Costura component reference. Overwrite if necessary.

These packages are used to embed the CI.ATFX.Reader.dll and Common.dll files to the exe file during build.



Then for the final components, remove them and reference the CI.ATFX.Reader.dll and Common.dll files in the ATFX API Package bin folder.

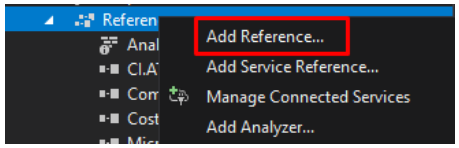


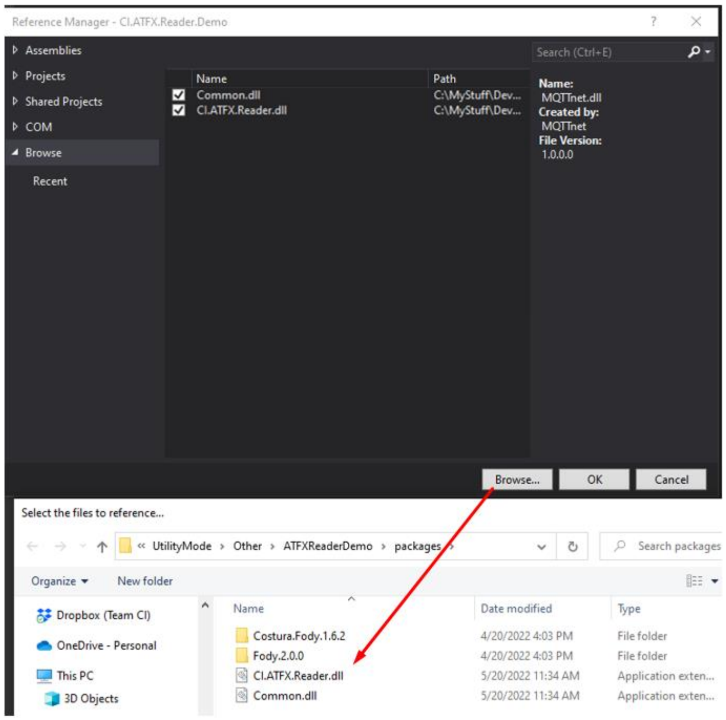
After doing all that, the project can now be built.

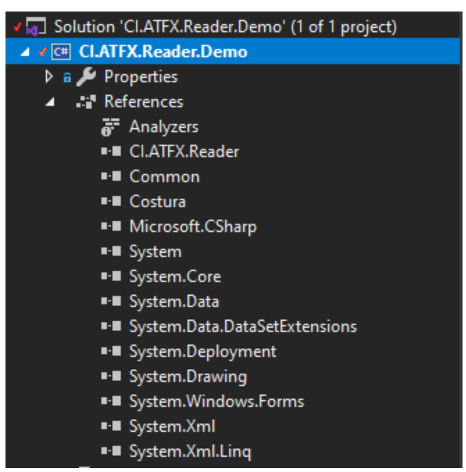
```
Build started...
1>------ Build started: Project: CI.ATFX.Reader.Demo, Configuration: Debug Any CPU ------
1>    Fody: Fody (version 2.0.0.0) Executing
1>      Fody/Costura:       No reference to 'Costura.dll' found. References not modified.
1>      Fody/Costura:       Embedding 'C:\Program Files\Crystal Instruments\Signal Reader API\bin\CI.ATFX.Reader.dll'
1>      Fody/Costura:       Embedding 'C:\Program Files\Crystal Instruments\Signal Reader API\bin\Common.dll'
1>    Fody:   Finished Fody 655ms.
1>    Fody:   Skipped Verifying assembly since it is disabled in configuration
1>    Fody:   Finished verification in 2ms.
1>  CI.ATFX.Reader.Demo -> C:\Users\KevinCheng\Downloads\ATFX API Package v1.4\ATFXReaderDemo\bin\Debug\CI.ATFX.Reader.Demo.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Importing and Referencing C# DLL Files

The C# Demo code has a Visual Studio project that can be opened to see how the C# DLL files are referenced in the project. The C# DLL files can be directly referenced into the project by right clicking References -> Add References -> Browse in Reference Manager window -> Locating the DLL files in ATFX API Package\bin folder.







After the C# DLL files have been referenced in the C# Demo, the ATFX API namespace can be imported to use the various classes and properties.

Below are several imports from the ATFX API that are used in the C# Demo code:

```
using EDM.RecordingInterface;
using EDM.Recording;
using ASAM.ODS.NVH;
using Common;
using Common.Spider;
using EDM.Utils;
```

The C# Demo project also comes with the Fody/Costura package that embeds any referenced dll files into the buildable exe file.

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>**	
○ 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)
○ Frame[0]	**<double[]>**	Array of x-values
○ Frame[1]	**<double[]>**	Array of y-values
○ 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.
GetFrame(int, _SpectrumScalingType, string)	Double[][]	Returns a **double[][]** with the data frame at that index. There are two additional parameters that can convert the returned data based on the spectrum type and the engineering unit. 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.

```xml
<files>
  <component>
    <identifier>External__{4499520}_REC_{20220419}(1)</identifier>
    <filename>{4499520}_REC_{20220419}(1).dat</filename>
  </component>
  <component>
    <identifier>External__{4499520}_REC_{20220419}(1)</identifier>
    <filename>{4499520}_REC_{20220419}(1).ts</filename>
  </component>
</files>
```

```xml
<files>
  <component>
    <identifier>External__REC0041</identifier>
    <filename>REC0041.dat</filename>
  </component>
  <component>
    <identifier>External__REC0041</identifier>
    <filename>REC0041.gps</filename>
  </component>
</files>
```

In order to extract the data from these types of files users will need to import **EDM.Utils**, which will allow access to **Utilty** class that offers various getter methods that return properties or lists of data from the ATFX file.

```
using EDM.Utils;
```

The **Utility** method to use to get external file components and return them as **IRecording** objects in a list is **GetListOfAllRecordings(IRecording)**. This method will at least return a list containing **one** IRecording object that is the main recording of the ATFX file and contains the bulk of the data.

```
private void ShowRecordings(IRecording rec)
{
  List<IRecording> recordingList = Utility.GetListOfAllRecordings(rec);
}
```

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.

The Utility method to use is **GetListOfAllSignals(IRecording)** that will return all the signals inside the passed in recording in a list. And if that recording contains .ts and \ or .gps file, it will also add their signals to the returned list.

```
private void ShowSignals(IRecording rec)
{
  List<ISignal> recordingList = Utility.GetListOfAllSignals(rec);
}
```

Opening a Time Stamp Signal (TS) or GPS Location File

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

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 the Utilty **GetListOfProperties** method, which will return a 2D list of strings. Each list contains the property name and property value.

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 Recording Property

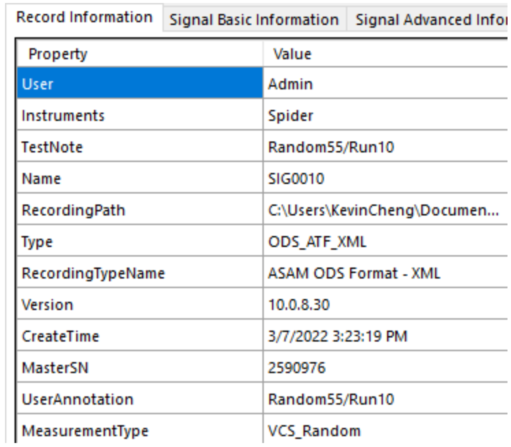
```
DateTime createTime = [IRecording object].RecordingProperty.CreateTime;

string instrument = [IRecording object].RecordingProperty.Instruments;

uint masterSN = [IRecording object].RecordingProperty.MasterSN;

etc.
```

GetListOfProperties

The Utility GetListOfProperties method is useful in getting a list of various data types in the RecordingProperty class. It returns a 2D list of strings with the property name and property value for each list.

```
Utility.GetListOfProperties(object item);
```

```
var recordingPath = "C:\Sig001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))
{
  foreach(List<string> property in Utility.GetListOfProperties(rec.RecordingProperty))
  {
    dataGridRecord.Rows.Add(property[0], property[1]);
  }
}
```



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 methods:

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(nvhEnvironment.TimeZoneString))
    {
      dgvRecInfo.Rows.Add("Time Zone", nvhEnvironment.TimeZoneString);
    }

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

    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 referenced in the link below:

https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=net-6.0#fields

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;
    DateTimeNano createTimeUTC;
    if (isLocal)
    {
      createTimeUTC = new DateTimeNano(nvhRec.RecordingProperty.CreateTime,
nvhMeasurement.NanoSecondElapsed);
    }
    else
    {
      createTimeUTC = new
DateTimeNano(Utils.GetUTCTime(nvhRec.RecordingProperty.CreateTime, null),
        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.TotalNanoSeconds);
    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 Input Channel Table Data

The Input Channel Table is a list of channels based on how many inputs of the test's recording instrument system, such as a Spider 80X 8 Channels. These channels, attached with sensors, measured physical quantities to voltages by the front-end hardware then read into physical units by the EDM software.

Below is a list of data columns that the input channel has for each channel:

Data Column Name	Description
On/Off	Enables or disables the channel.
Location ID	Assigns a custom label used to identify the source in the signal display and other setup windows.
Measurement Quantity	Defines the physical unit that will be measured by the sensor connected to the channel.
Sensitivity	Sets the proportionality factor for the measurement (millivolts per engineering unit) given as a parameter of the sensor.
Input Mode	The electrical interface mode of the sensor. **DC-Differential** - Neither of the input connections is referenced to the local ground. The input is taken as the potential difference between the two input terminals, and any potential in common with both terminals is canceled out.

	DC-Single End - One of the input terminals is grounded and the input is taken as the potential difference of the center terminal with respect to this ground. Use this mode when the input needs to be grounded to reduce EMI noise or static buildup. **AC-Differential** - A differential input mode that applies a low-frequency high-pass (DC-blocking) analog filter to the input. It rejects common mode signals and DC components in the input signal. **AC-Single End** - Grounds one of the input terminals and enables the DC-blocking analog filter. **Integral Electronic PiezoElectric (IEPE (ICP))** - A class of transducers that are packaged with built-in voltage amplifiers powered by a constant current. **Charge** - For high-sensitivity piezoelectric units that lack a built-in voltage mode amplifier (i.e. IEPE), allowing them to be used in high-temperature environments.
Input Range	The voltage range of the Input Mode.
Sensor	Defines the sensor setting applied to an input channel.
Max Sensor Range	Defines the maximum input voltage allowed.
Integration	Allows having No Integration, Integration, or Double Integration applied.
High-Pass Filter Fc (Hz)	Sets the digital high-pass filter frequency, used to block spurious low frequency and DC signals. To measure very low frequency or DC signals set this value to zero and use the DC-SE or the DC-DI input mode.
Channel Type	The type of channel, whether it is a Control or Monitor channel.
Measurement Point	The measure point that the input channel is connected to.
DOFs	The degree of freedom of the channel that is the combination of entered Measurement Point and Coordinate.
Control Weighting	Used when more than one control channel is present for weighted averaging. See the description for the Control Strategy test parameter. The weighting factors are automatically normalized. For example, enter weighting factor 2.0 for channel 1, 1.0 for channel will be the same as entering factor 4.0 for channel 1 and 2.0 for channel 2.
Description	Used to add users' notes.
Coordinate	Specifies the measurement position and direction of the sensor.
Time Weighting	Defines the time weighting for exponential averaging. (Only available in acoustic test)

Reading the Input Channel Data Through Utility Class

To read the Input Channel Table data stored in the ATFX file, it is extracted from the IRecording object using the Utility **GetChannelTable** method, which will return a 2D list of strings. Each list contains one row of channel data.

```
Utility.GetChannelTable(IRecording);
```

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

  foreach (List<string> channel in Utility.GetChannelTable(rec))
  {
    dgvChannel.Rows.Add(channel.ToArray());
  }
}
```

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

Calling Individual Properties of Input Channel

It is possible to directly call input channel data from an IRecording object, although it is recommended to use the Utility GetChannelTable method. To get the necessary input channel object, the IRecording must be converted to a **ODSNVHATFXMLRecording** object to locate the **ChnSensitivitys** property. This property 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;
```

Below shows a way of extracting data directly from the NVHTestEquipmentPart object.

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

```
  ODSNVHATFXMLRecording odsRec = rec as ODSNVHATFXMLRecording;

  foreach (ChannelSensitivity eq in odsRec.ChnSensitivitys)
  {
    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);
  }
}
```

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 Utilty **GetListOfProperties** method, which will return a 2D list of strings. Each list contains the property name and property value.

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

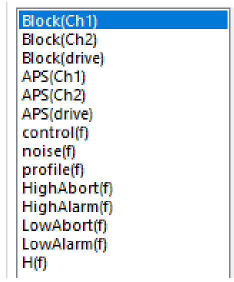
Name	Return Type	Descriptions
GetFrame(int)	Double[][]	Returns a **double[][]** with the data frame at that index A snapshot of measurement data consisting of X, Y and sometimes Z values.
GetFrame(int, _SpectrumScalingType, string)	Double[][]	Returns a **double[][]** with the data frame at that index. There are two additional parameters that can convert the returned data based on the spectrum type and the engineering unit. 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. This can be done by the Utility **GetListOfAllSignals** that returns a list of ISignal from the ATFX file.

```
Utility.GetListOfAllSignals(IRecording);
```

```
var recordingPath = “C:\Sig001.atfx”;
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))
{
  lbSignalDataInfo.Items.AddRange(Utility.GetListOfAllSignals(rec).ToArray());
}
```



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.
```

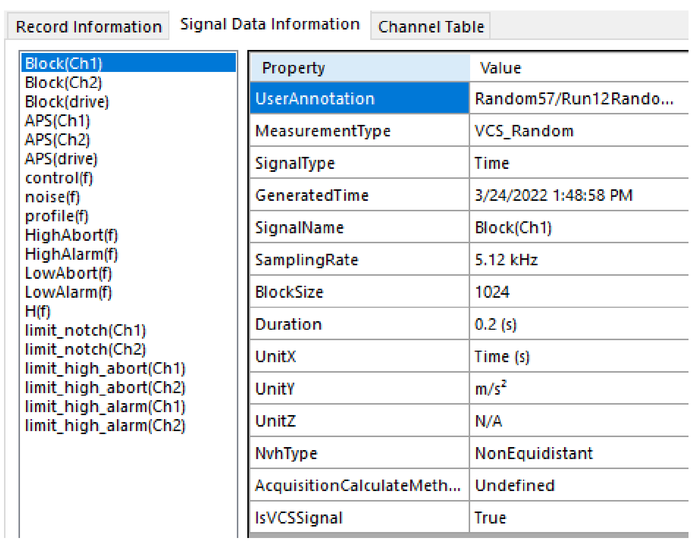
GetListOfProperties

The Utility GetListOfProperties method is useful in getting a list of various data types in the SignalProperties class. It returns a 2D list of strings with the property name and property value for each list.

The following code snippets display the signal information.

```
Utility.GetListOfProperties(object item);
```

```
private void BtnSignalBasicInfo_Click(object sender, EventArgs e)
{
  if (lbSignalDataInfo.SelectedItem is ISignal signal)
  {
    foreach(List<string> property in Utility.GetListOfProperties(signal.Properties))
    {
      dgvSignalDataInfo.Rows.Add(property[0], property[1]);
    }
  }
}
```



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/s2) when data saved
controlRMS	double	current control RMS (m/s2) 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/s2) when data saved
targetRMS	double	target RMS (m/s2) 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.
```

GetListOfProperties

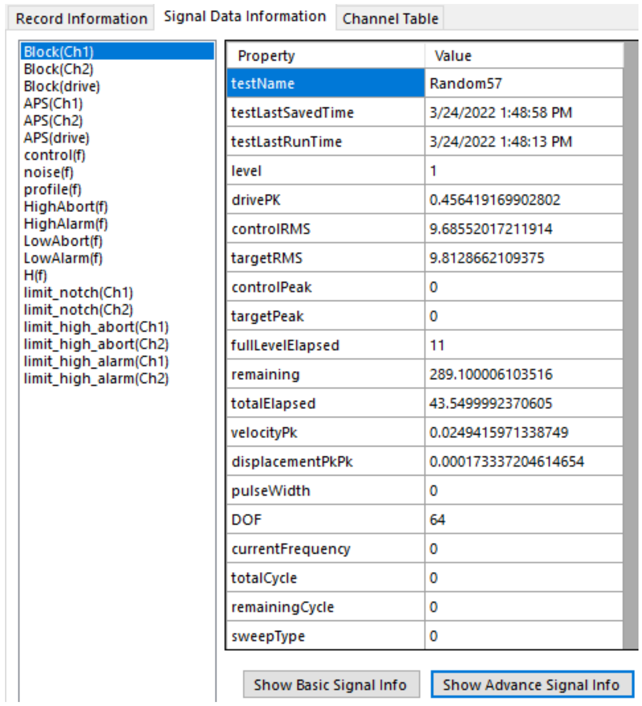
Here is a code snippet for displaying the advance signal information, depending on if the signal comes from VCS or DSA.

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

```
Utility.GetListOfProperties(object item, bool showPublicField);
```

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

  foreach(List<string> property in Utility.GetListOfProperties(item, showPublicField))
  {
    grid.Rows.Add(property[0], property[1]);
  }
}
```

```
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);
    }
  }
}
```



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.TotalNanoSeconds;
int seconds = signal.Properties.GeneratedTime.Second;
etc.
```

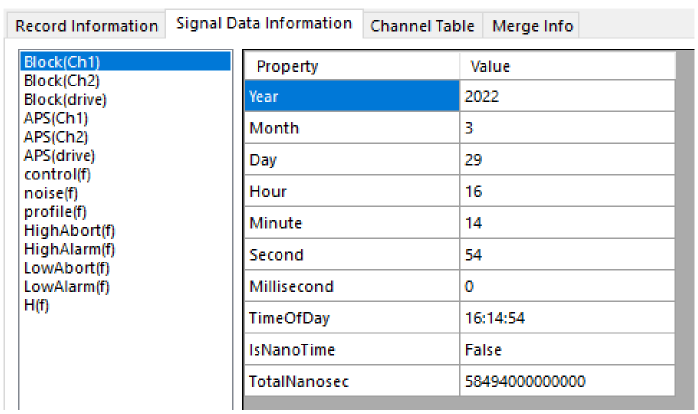
GetListOfProperties

The Utility GetListOfProperties method is useful in getting a list of various data types in the DateTimeNano class.

```
Utility.GetListOfProperties(object item);
```

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

```
private void BtnShowGeneratedTime_Click(object sender, EventArgs e)
{
  if (lbSignalDataInfo.SelectedItem is ISignal signal)
  {
    foreach(List<string> property in
Utility.GetListOfProperties(signal.Properties.GeneratedTime))
    {
      dgvSignalDataInfo.Rows.Add(property[0], property[1]);
    }
  }
}
```



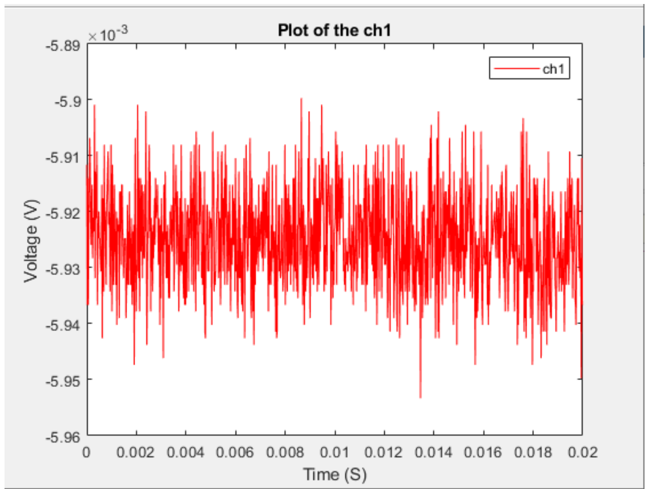
Reading the Data Values of a Signal Frame

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 to the following structure:

Concept	Class Type	Example
Signal	**<ISignal>**	Block(Ch1)
- Frame	**<double[][]>**	Signal.GetFrame(0)
○ Frame[0]	**<double[]>**	Array of x-values
○ Frame[1]	**<double[]>**	Array of y-values
○ 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

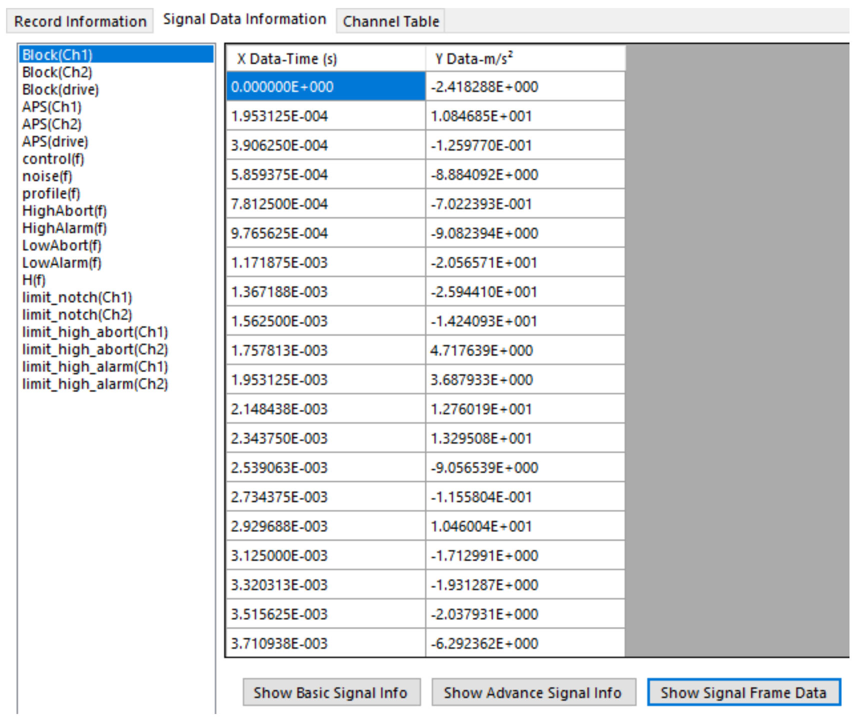
		domain signals, Properties refer to FrequencyDomainSignalProperties.
Recording	IRecording	Get the signal recording
Type	SignalType	Get the signal type, time/frequency domain Unknown 0 Time 1 Frequency 2 Trend 3

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.
GetFrame(int, _SpectrumScalingType, string)	Double[][]	Returns a **double[][]** with the data frame at that index. There are two additional parameters that can convert the returned data based on the spectrum type and the engineering unit. 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;
```



Reading Frequency Signal Frame Data

The ATFX API can read the frequency signal frame data in other spectrum types and engineering units. **Spectrum Type** defines the units for spectrum signals as power spectral density (EU^2/Hz), energy spectral density (EU^2s/Hz), squared units ($EU_{rms}{}^2$), peak units (EU_{peak}), or RMS (EU_{rms}).

The engineering units from EDM global settings should be saved in the ATFX file, however, the spectrum type is not. The **default** for the spectrum type is **(EUrms)^2**. Thus, if the data read by the ATFX API is different then what is in EDM, try passing in different engineering units and spectrum types.

Frequency Response Function (FRF) related signals, such as FRF, H, Cross Power Spectrum (CPS) and Fast Fourier Transform Spectral Analysis linear (FFT) spectrum are read in Real &

Imaginary. These signals also pair the Real & Imaginary numbers in the Y data, thus X data frame size may be 512 and the Y data frame size is 1024.

The ISignal class comes with a **GetFrame(int index, _SpectrumScalingType spectrum, string engineeringUnit)** that users can use to convert the returned frame data. And for reading the Y labels for the FRF related signals, the ISignal class has **GetYLabel**, which returns a list of strings. And depending on the signal, the first string in the list will be enough for the Y data label, but if it's a FRF related signal, the second string in the list will act as the imaginary type Y data label.

Note that spectrum types only apply to Power Spectrum and Linear Spectrum signals and do not apply to transfer functions, phase functions or coherence functions. Whereas the engineering units should change every signal. There are also spectrum signals that only has a select amount of spectrum types, such as Sine spectrum with EUrms, EUPeak and EUPeak-Peak or Octave spectrum with $EUrms^2$ and EUrms.

```
ISignal.GetFrame(int, _SpectrumScalingType, string);
```

```
ISignal.GetYLabel();
```

```
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 1 signal, select the signal whose name is ‘APS(Ch1)’
ISignal signalCh1 = signals.Where(sig => sig.Name == ‘APS(Ch1)').First();

// Get the frame, which is formatted like [[x1, x2, x3…], [y1, y2, y3…],…]
double[][] frame = signalCh1.GetFrame(0, _SpectrumScalingType.EUPeak,
AccelerationUnitEnumString.ArrayString[AccelerationUnitType.g]);
double[] xValues = frame[0];
double[] yValues = frame[1];

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

string signalCh1YLabel = signalCh1.GetYLabel()[0];

// If statement for obtaining the 2nd Y data label if the signal is related to FRF
// Also applies to Cross power spectrum and FFT
if (signal.Properties.NvhType == _NVHType.FrequencyResponseSpectrum)
{
  string signalCh1_2ndYLabel = signalCh1.GetYLabel()[1];
}
```

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)

X Data-Frequency (Hz)	Y Data- g (0-peak)
0	0.000316271071551983
25	0.000158114866204864
50	6.52407611600791E-06
75	5.56056284750977E-06
100	4.81845486436046E-06
125	4.27552921666523E-06
150	3.64940075677389E-06
175	3.59262165824685E-06
200	2.85317043372338E-06
225	2.85368845865449E-06
250	3.07391155226024E-06

Change how the signal frame data is read.
This does not change the values inside the ATFX file.
EUpeak g

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)

X Data-Frequency (Hz)	Y Data- $(m/s^2)^2$ (RMS)
0	4.80983629822731E-06
25	1.20214475318789E-06
50	2.04667740035802E-09
75	1.48678736877628E-09
100	1.11641829789733E-09
125	8.79004528542282E-10
150	6.40404641671921E-10
175	6.20632226855378E-10
200	3.91441426472738E-10
225	3.91583580494625E-10
250	4.54353721579537E-10

Change how the signal frame data is read.
This does not change the values inside the ATFX file.
$(EUrms)^2$ m/s^2

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
Block(Ch6)
Block(Ch7)
Block(Ch8)
Block(drive)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)
APS(Ch6)
APS(Ch7)
APS(Ch8)
APS(drive)
control(f)
noise(f)
profile(f)
HighAbort(f)
HighAlarm(f)
LowAbort(f)
LowAlarm(f)
H(f)
FRF(Ch2,Ch1)
FRF(Ch3,Ch1)

X Data-Frequency (Hz)	Y Data-Real $(m)/(m/s^2)$	Y data-Imaginary $(m)/(m/s^2)$
0	2.17429424083093E-05	0
5	-3.58616807716317E-06	9.93081448541488E-06
10	-2.79689572835196E-07	6.38803328456561E-07
15	-1.08067453652438E-07	8.15487837257933E-08
20	-2.81281273828426E-08	8.01220867430175E-09
25	-6.75226674573537E-09	-4.5313473862052E-08
30	3.39580319419497E-09	-5.8946718617392E-09
35	7.18608195171555E-09	4.75262886823202E-08
40	1.27004264882657E-08	4.73981494053533E-08
45	5.7980775736155E-09	1.14014326868528E-08
50	3.07961940393398E-08	-6.5433223284117E-09
55	1.10163531630292E-08	-2.2099783336671E-08
60	1.75667995705453E-08	1.49446020003552E-08
65	3.95185537627185E-08	3.55755886971565E-08
70	1.47341427947367E-08	1.7456903478319E-08

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
Block(Ch6)
Block(Ch7)
Block(Ch8)
Block(drive)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)
APS(Ch6)
APS(Ch7)
APS(Ch8)
APS(drive)
control(f)
noise(f)
profile(f)
HighAbort(f)
HighAlarm(f)
LowAbort(f)
LowAlarm(f)
H(f)
FRF(Ch2,Ch1)
FRF(Ch3,Ch1)
H(Ch2,Ch1)
H(Ch3,Ch1)
H(Ch1,Ch2)
H(Ch3,Ch2)
H(Ch1,Ch3)

X Data-Frequency (Hz)	Y Data-Real $(m)/(m/s^2)$	Y data-Imaginary $(m)/(m/s^2)$
0	1.18312773338403E-05	0
4.99999992549419	-2.42805367633991E-06	1.55340148921823E-05
9.99999985098838	-1.42553767545905E-06	-7.00795908414875E-07
14.9999997764826	-2.05770874117661E-07	2.50104761789771E-07
19.9999997019768	-4.31938680378607E-08	1.89972055864018E-08
24.9999996274709	5.92527860110437E-10	-4.0975738357929E-08
29.9999995529651	6.78589806568652E-09	-3.50743505350692E-08
34.9999994784593	5.12031572696969E-09	2.32202221894795E-08
39.9999994039535	2.54867309479323E-08	4.87005671345742E-08
44.9999993294477	8.84821993452078E-09	1.72598184633443E-08
49.9999992549419	3.7229622051882E-08	-1.24026939829491E-08
54.9999991804361	1.45628975545264E-09	-2.90616544162958E-08
59.9999991059303	5.26843857429071E-09	1.52049284274369E-09
64.9999990314245	2.67518469598826E-08	2.68895021804383E-08
69.9999989569187	6.54148779588581E-09	2.88359309763564E-08
74.9999988824128	2.90316237716581E-09	3.83329350484019E-08
79.999998807907	1.54965622556347E-08	5.24112486743888E-08
84.9999987334012	5.89037041365259E-09	2.11882795753127E-08

Getting Spectrum Types or Engineering Units

Each signal is a specific type that has its own spectrum type and engineering unit (EU) that can convert the frame data when passing it through the GetFrame method.

For example:

APS signal in Acceleration

Spectrum Type: $EUrms^2$, EUrms, EUPeak, EUPeak-Peak, EU^2/Hz, EU^2s/Hz, sqrt(EU^2/Hz), sqrt(EU^2s/Hz)

Acceleration EU: m/s^2, cm/s^2, mm/s^2, g, ft/s^2, in/s^2, mil/s^2, gal

The Utility class has several methods for getting the enum _SpectrumScalingType, the spectrum type names, and the engineering unit names.

Name	Return Type	Descriptions
GetListOfSpectrumTypes	List<string>	Takes in a ISignal and returns a list of strings of spectrum type names depending on the signal NVH type.
GetSpectrumType	_SpectrumScalingType	Takes in a string that is the spectrum type name and returns the equlivant enum _SpectrumScalingType.
GetSpectrumTypeString	string	Takes in a _SpectrumScalingType and returns the equlivant string spectrum type name.
GetSignalQuantityEngiUnitStrings	string[]	Takes in a ISignal and returns a string array that contain engineering units of a signal quantity.

```
Utility.GetListOfSpectrumTypes(ISignal);
```

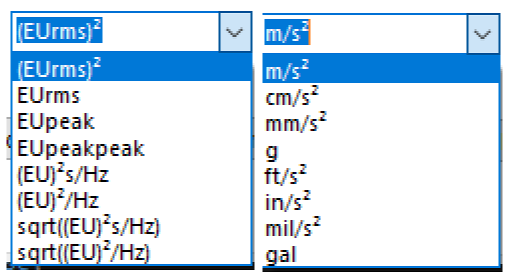
```
Utility.GetSpectrumType(string);
```

```
Utility.GetSpectrumTypeString(_SpectrumScalingType);
```

```
Utility.GetSignalQuantityEngiUnitStrings(ISignal);
```

```
private void LbSignalDataInfo_SelectedIndexChanged(object sender, EventArgs e)
{
  if (lbSignalDataInfo.SelectedItem is ISignal signal)
  {
    if (signal.Type == SignalType.Frequency &&
        (signal.Properties.NvhType == _NVHType.FrequencyResponseSpectrum ||
         signal.Properties.NvhType == _NVHType.CrosspowerSpectrum ||
         signal.Properties.NvhType == _NVHType.Coherence ||
         signal.Properties.NvhType == _NVHType.Equidistant))
    {
      cbEngiUnit.Items.Clear();
```

```
        cbEngiUnit.Enabled = false;
      }
      else
      {
        cbEngiUnit.Enabled = true;

        cbEngiUnit.Items.Clear();
        cbEngiUnit.Items.AddRange(Utility.GetSignalQuantityEngiUnitStrings(signal));
        cbEngiUnit.SelectedItem = signal.GetUnit(1);
      }

      if (signal.Type == SignalType.Frequency && !signal.Name.Contains("Swept THD") &&
          (signal.Properties.NvhType == _NVHType.AutopowerSpectrum ||
           signal.Properties.NvhType == _NVHType.OctaveAutopowerSpectrum ||
           signal.Properties.NvhType == _NVHType.OrderAutopowerSpectrum))
      {
        cbSpecScaleType.Enabled = true;

        cbSpecScaleType.Items.Clear();

        cbSpecScaleType.Items.AddRange(Utility.GetListOfSpectrumTypes(signal).ToArray());
        cbSpecScaleType.SelectedItem =
Utility.GetSpectrumTypeString(signal.Properties.specType);
      }
      else
      {
        cbSpecScaleType.Items.Clear();
        cbSpecScaleType.Enabled = false;
      }
    }
}
```



Reading NVH Test Configuration Parameters

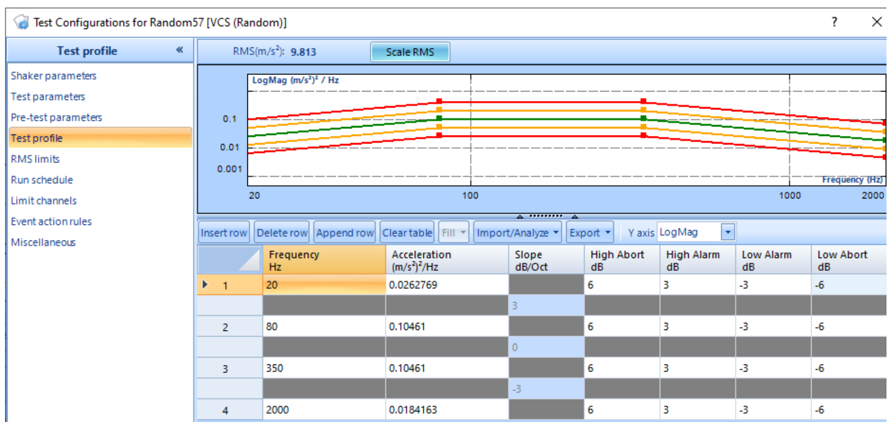
A Noise, Vibration and Harshness (NVH) Parameter Set is a set of parameter keys that a signal stores information regarding the signal properties, recording properties and testing configuration parameters. For the list of parameter keys and their descriptions, refer to the **Property Glossary – NVHParameterSset Parameter Keys** section.

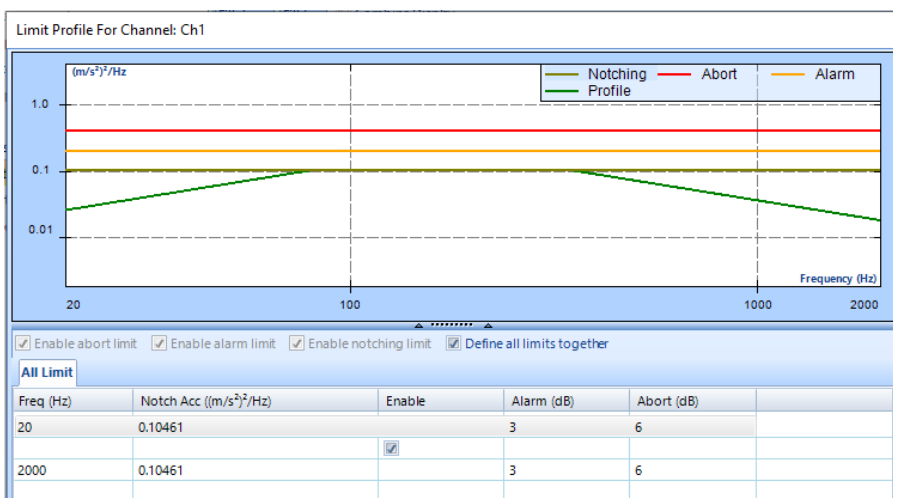
For the complete list of fields in NVHParameterSet, it is recommended to find these fields in the File Reader API for CI Measurement Data Class Methods.chm file under ASAM.ODS.NVH -> NVHParameterSet Class -> NVHParameterSet Fields.

To read the NVH Parameter Set stored in a ATFX file, each signal can get a NVH Test Configuration Parameter value and type through the Utility **GetSignalNVHParameter** or **GetSignalProfileOrLimit** with a **NVHParameterSet** parameter key. Most signals share the same testing configuration parameter values.

The GetSignalNVHParameter returns a list of strings that contains the signal parameter data type and the parameter value.

For certain signal parameters such as the **Test Profile** or **Channel Limit Profile**, the GetSignalProfileOrLimit method is used to return a 2D list of strings where each list contains a row of data.





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;
```

Reading a Signal NVH Parameter Key

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

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

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

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

etc.
```

Reading a Signal NVH 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 a List of NVH Parameter Keys Through Utility Class

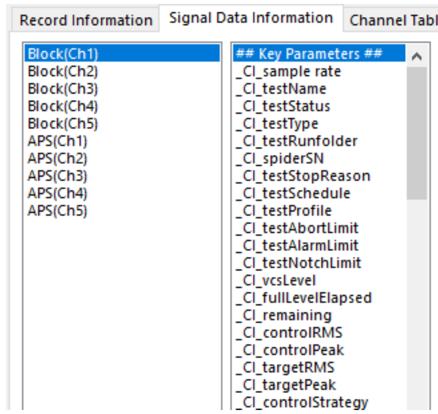
Given that there is a list of parameters for each signal, it would be better to store the list of parameters into another list object for the user interface and other means of accessing the data.

The Utility **GetListOfNVHParameterSet** returns a list of strings with empty headers to easily look through the list. The list will also have important parameters placed first and then the rest of the NVHParameterSet keys.

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.

```
Utility.GetListOfNVHParameterSet();
```

```
var recordingPath = “C:\Sig001.atfx”;
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))
{
  lbSignalDataInfo.Items.AddRange(Utility.GetListOfAllSignals(rec).ToArray());
  lbSignalParameters.Items.AddRange(Utility.GetListOfNVHParameterSet().ToArray());

  if (lbSignalParameters.Items.Count > 0)
    lbSignalParameters.SelectedIndex = 0;
}
```



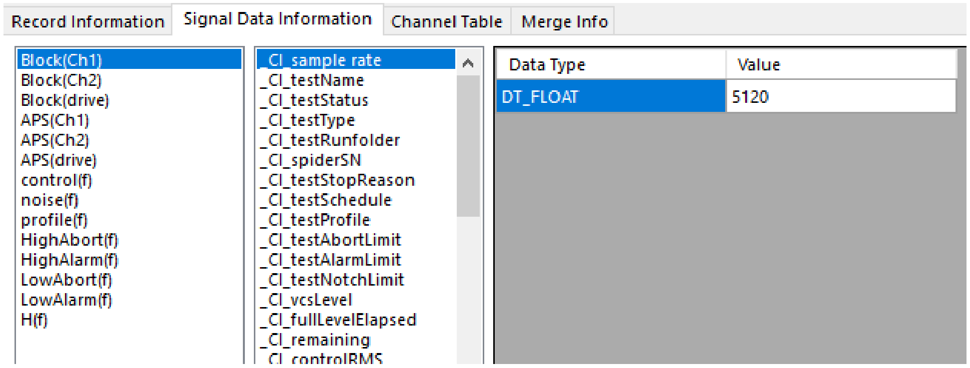
Reading a NVH Parameter Key & Type Through Utility Class

```
Utility.GetSignalProfileOrLimit(ISignal sig, string parameterKey);
```

```
Utility.GetSignalNVHParameter(ISignal sig, string parameterKey);
```

```
private void ShowParameters(DataGridView grid, ISignal sig, string parameterKey)
{
  grid.Rows.Clear();

  if (parameterKey == NVHParameterSet.testProfile)
  {
    foreach (List<string> entry in Utility.GetSignalProfileOrLimit(sig, parameterKey))
    {
      grid.Rows.Add(entry.ToArray());
    }
  }
  else if (parameterKey == NVHParameterSet.testAbortLimit ||
           parameterKey == NVHParameterSet.testAlarmLimit ||
           parameterKey == NVHParameterSet.testNotchLimit)
  {
    foreach (List<string> entry in Utility.GetSignalProfileOrLimit(sig, parameterKey))
    {
      grid.Rows.Add(entry.ToArray());
    }
  }
  else
  {
    List<string> signalParam = Utility.GetSignalNVHParameter(sig, parameterKey);
    grid.Rows.Add(signalParam.ToArray());
  }
}
```

```
private void BtnSignalParam_Click(object sender, EventArgs e)
{
  string parameterKey = lbSignalParameters.SelectedItem as string;
  if (lbSignalDataInfo.SelectedItem is ISignal signal &&
      !string.IsNullOrEmpty(parameterKey))
  {
    ShowParameters(dgvSignalDataInfo, signal, parameterKey);
  }
}
```



Reading Merged Information

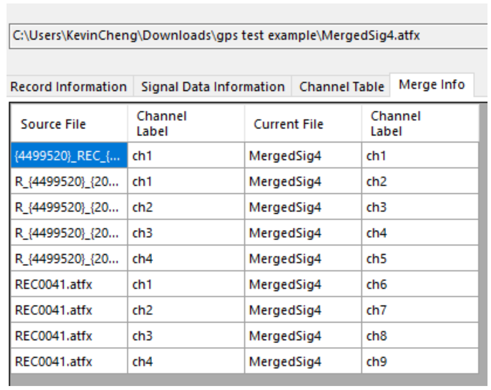
Depending on the ATFX file, it can contain multiple other atfx files. It is still converted into a singular IRecording object with the RecordingManager OpenRecording. Then the Utility **GetMergeInfo** is used to return a 2D list of strings, where each list contains data regarding each ATFX file channels. It also **output** an **int** that is the number of ATFX files in the merged ATFX file.

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

```
Utility.GetMergeInfo(IRecording, out int sigMapCount);
```

```
private void ShowMergeInfo(IRecording rec)
{
  try
  {
    dgvMergeInfo.SuspendLayout();
    dgvMergeInfo.Rows.Clear();

    List<List<string>> mergeInfo = Utility.GetMergeInfo(rec, out int sigMapCount);

    if (sigMapCount == 0)
    {
      dgvMergeInfo.Columns[0].Visible = false;
      dgvMergeInfo.Columns[1].Visible = false;
    }
    else
    {
      dgvMergeInfo.Columns[0].Visible = true;
      dgvMergeInfo.Columns[1].Visible = true;
    }

    foreach (List<string> merge in mergeInfo)
    {
      dgvMergeInfo.Rows.Add(merge.ToArray());
    }
    this.Refresh();
  }
  finally
  {
```

```
      dgvMergeInfo.ResumeLayout();
      dgvMergeInfo.PerformLayout();
    }
  }
```



ATFX API Method List

The following section is a short preview of the various classes and interfaces in the API. For a more detailed view, please refer to the File Reader API for CI Measurement Data Class Methods.chm file.

List of Available Modules

Module	Descriptions
Recording Manager	Provide methods to manage/operate Recording Objects, e.g. open or close Recording Objects
ODS Recording	Provide methods to access properties of Recording Objects
ODS Signal	Provide methods to access properties of Signal Objects
DateTimeNano	Provide methods to create a DateTimeNano object with similarities to DateTime but with more accuracy up to nanoseconds.
Utility	Provide methods to easily get data from the ATFX file without having to understand the complexity of ASAM ODS source code

Recording Objects refer to files recorded/saved in EDM.

Signal Objects refer to signals included in recording objects.

Recording Manager Module

Name to Be Called	Type	Descriptions
OpenRecording	Method	Open the file
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	Attribute	Signal Name
Type	Attribute	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(Utils.GetUTCTime
(nvhRec.RecordingProperty.CreateTime, null), 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.TotalNanoSeconds);
  Console.WriteLine(createTimeUTC.ToNanoString());
}
```

Utility Module

Name to Be Called	Type	Descriptions
GetListOfAllRecordings	Method	Takes in a IRecording and returns a List<string> that contains all available recordings in a ATFX file.
GetListOfAllSignals	Method	Takes in a IRecording and returns a List<string> that contains all available signals in a ATFX file.
GetListOfNVHParameterSet	Method	Returns a List<string> that contains all available NVHParameterSet keys and some

		empty header strings for categories and easier to look through.
GetListOfProperties	Method	Takes in an object and bool and returns a 2D List<string> where each list contains the property name and property value.
GetChannelTable	Method	Takes in a IRecording and returns a 2D List<string> where each list contains a channel row.
GetSignalNVHParameter	Method	Takes in a ISignal and string and returns a List<string> that contains the parameter data type and parameter value.
GetSignalProfileOrLimit	Method	Takes in a ISignal and string and returns a 2D List<string> where each list contains a row of a test profile.
GetMergeInfo	Method	Takes in a IRecording and returns an int count of how many ATFX files in the merged ATFX file. And a 2D List<string> where each list contains data regarding each ATFX file channel.
GetListOfSpectrumTypes	Method	Takes in a ISignal and returns a list of strings of spectrum type names depending on the signal NVH type.
GetSpectrumType	Method	Takes in a string that is the spectrum type name and returns the equlivant enum _SpectrumScalingType.
GetSpectrumTypeString	Method	Takes in a _SpectrumScalingType and returns the equlivant string spectrum type name.
GetSignalQuantityEngiUnitStrings	Method	Takes in a ISignal and returns a string array that contain engineering units of a signal quantity.

Property Glossary

The following properties and methods can be found in the chm file and are listed here for a quicker reference and to highlight the most important properties and methods 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. ATFX, 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 deprived from the ISignal GetParameter<T> and GetParameterType where the methods gets the the value and data type of each parameter key.

Parameter Key	Type	Description
`abortSensitivity`	float	Defines the threshold for when an abort is called, based on several independent criteria
`average`	long	Number of blocks that are averaged for the control spectrum
`averageMode`	long	The method of averaging tests over blocks
`averageNumber`	long	The number of blocks that are ensemble averaged for the signal spectrum
`bandWidth`	float	Bandwidth of the proportional filter
`blockT`	float	Duration of time for the block
`blockTSize`	string	Duration of time for the block over block size
`controlPeak`	double	Control peak (m/s2) when data saved
`controlRMS`	double	Current control RMS (m/s2) when data saved

`controlStrategy`	string	Determines whether one or multiple control channels are used, and how the composite control signal is generated
`currentFrequency`	float	Current frequency when data saved (Sine)
`deltaF`	double	Delta Frequency
`deltaFreq`	string	Known as the frequency resolution, this sets the spacing between spectral frequency lines
`deltaT`	float	Delta Time
`displacementPkPk`	double	Displacement peak peak (m) when data saved
`DOF`	long	Degree Of Freedom
`driveLimit`	float	Limits the absolute maximum voltage output of the drive signal during the schedule test
`drivePK`	double	Current drive peak (voltage) when data saved
`fftAverageOnOff`	long	Whether the test uses FFT average or not
`filterType`	long	Determine how the filter bandwidth is changing and the bandwidth
`frequencyRange`	double	The maximum frequency resolved by the FFT transform by adjusting the sample rate
`fullLevelElapsed`	double	Time since full level has elapsed in seconds Ex. 636.2
`highRPM`	float	High end of RPM
`initialDrive`	float	The initial peak voltage of the drive signal that is set before it ramps up
`intervalBetweenPulses`	double	The time period between successive pulses
`lines`	string	Number of spectral lines, proportional to block size
`lowRPM`	float	Low end of RPM
`maximumDrive`	double	A safety limit set to protect the shaker during sine ramping up and pre-test process
`measureStrategy`	string	Defines how the sine waves are measured
`overlapRatio`	string	Determines what proportion of each time block is overlapped with the previous block when calculated the FFT
`remaining`	double	Time remaining in test schedule in seconds Ex. 299
`sampleRate`	float	Number of data samples acquired per second

		Ex. 5120
`sigmaClipping`	float	Limits the peaks of the output voltage distribution based on a factor of Sigma
`signalPlotPoints`	long	The number of frequency lines of the displaying spectrum
`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)
`sweepType`	string	Determine how the signal plot points are distributed across the frequency axis
`targetPeak`	double	Target peak (m/s2) when data saved
`targetRMS`	double	Target RMS (m/s2) 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. 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
`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.

Method	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

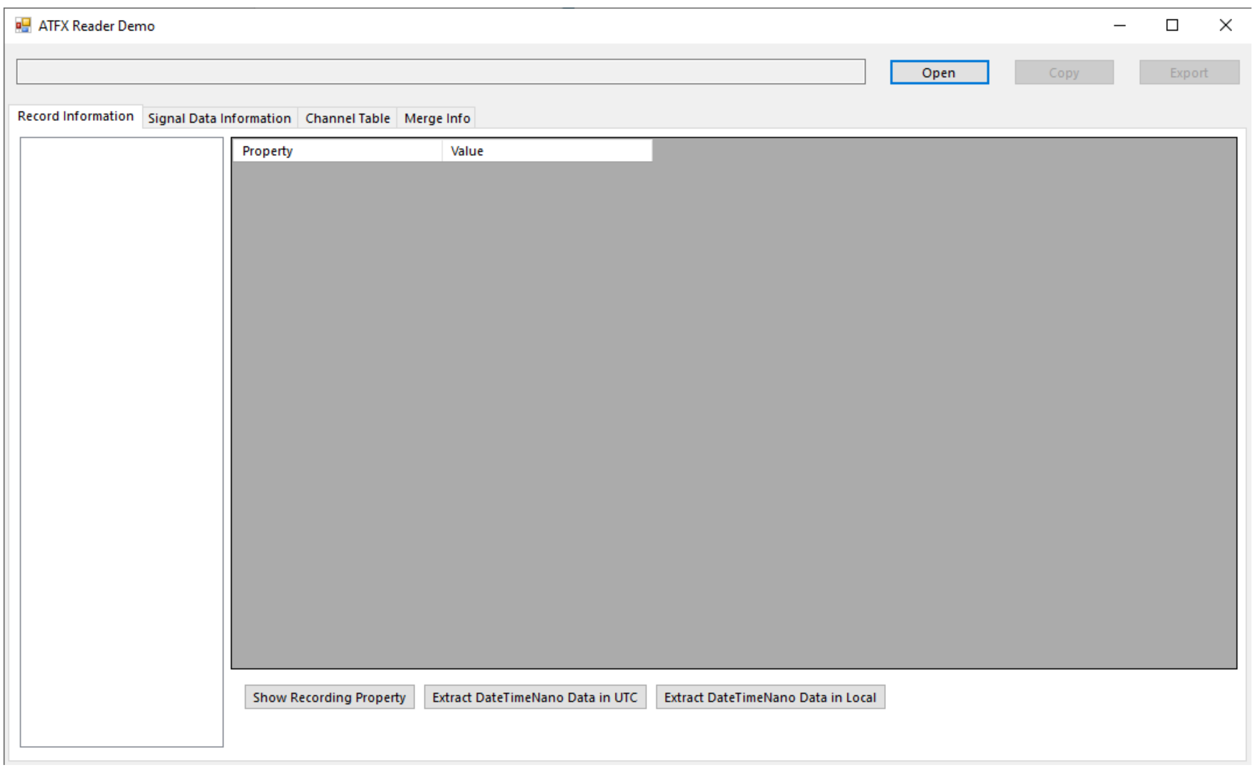
ATFX API Coding Languages

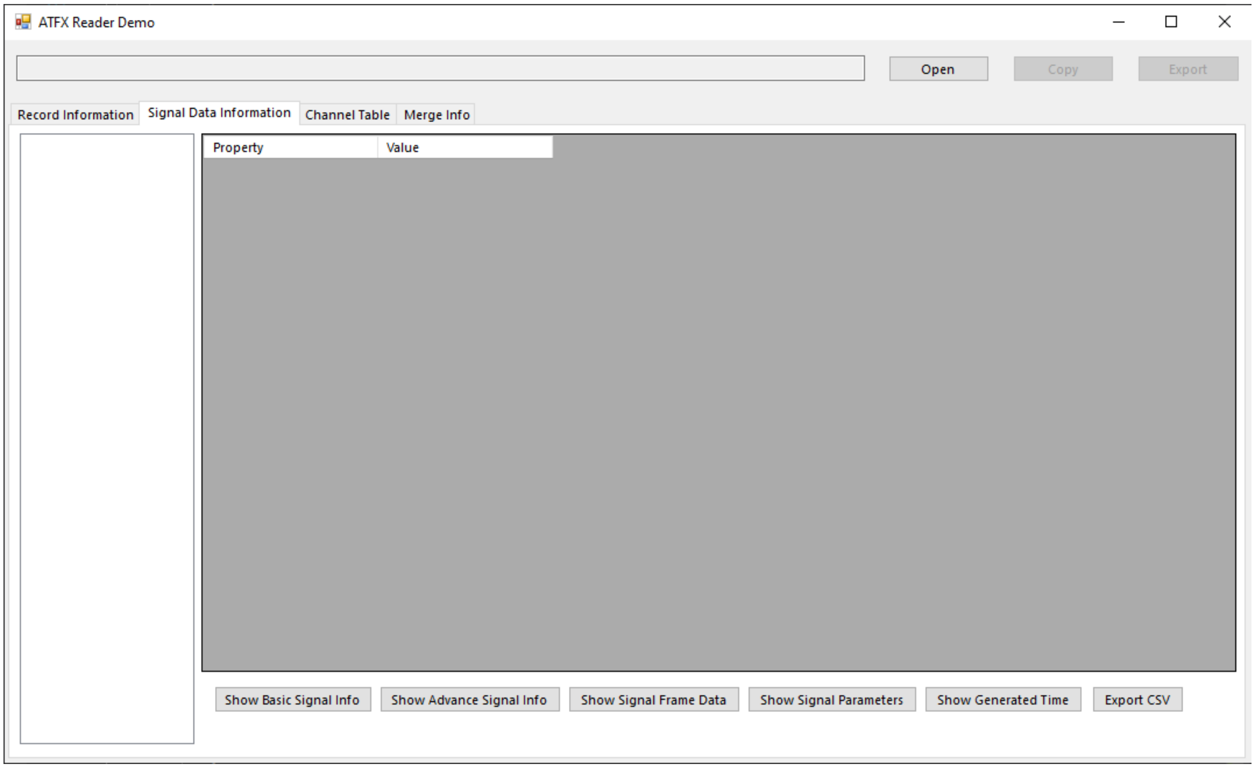
The ATFX API have C# DLL files that are used with the C# language, but there are ways to use the DLL files for other languages such as Python, LabVIEW and Matlab. The following section will demostrate how to import the DLL files and how to call the methods and properties.

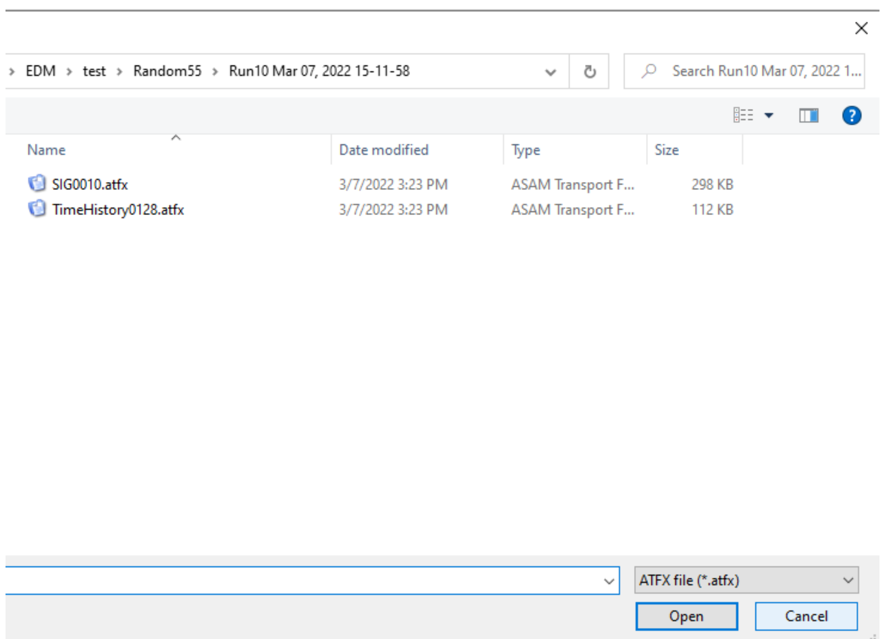
C# Demo Program

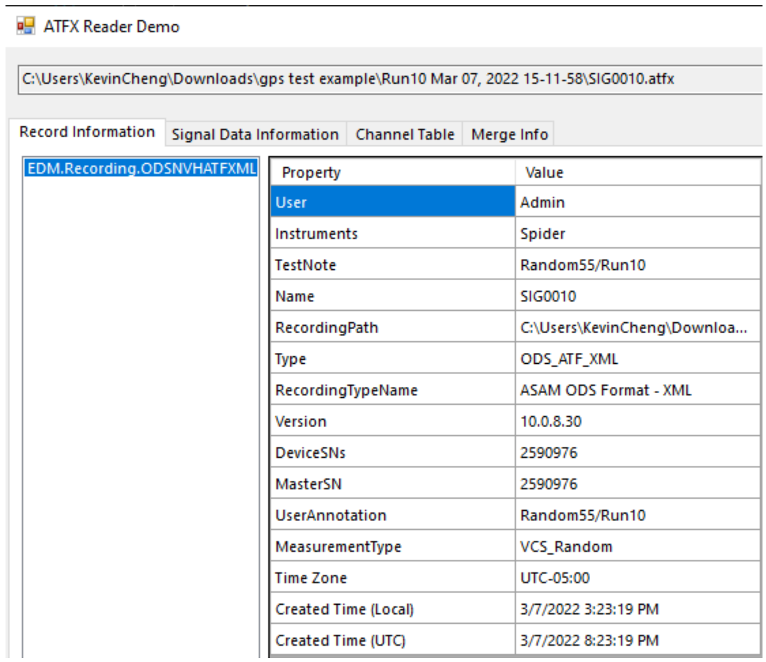
This is a demo program that demonstrates the API with a user interface that opens and displays the data stored in a ATFX file for the user to see. Instructions to how to import the DLL files and how to call the methods and properties are listed in the **API C# Demo Examples**.

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




ATFX Reader Demo
Open
Copy
Export
Record Information
Signal Data Information
Channel Table
Merge Info
Property
Value
Show Basic Signal Info
Show Advance Signal Info
Show Signal Frame Data
Show Signal Parameters
Show Generated Time
Export CSV

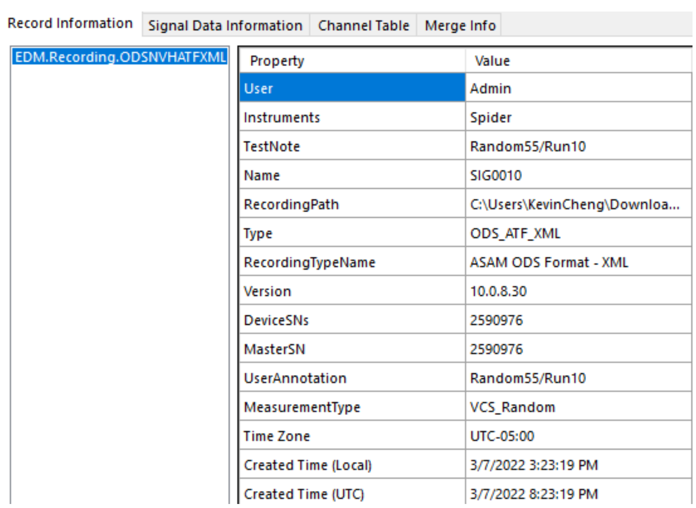

EDM › test › Random55 › Run10 Mar 07, 2022 15-11-58
Search Run10 Mar 07, 2022 1...
Name
Date modified
Type
Size
SIG0010.atfx 3/7/2022 3:23 PM ASAM Transport F... 298 KB
TimeHistory0128.atfx 3/7/2022 3:23 PM ASAM Transport F... 112 KB
ATFX file (*.atfx)
Open
Cancel



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

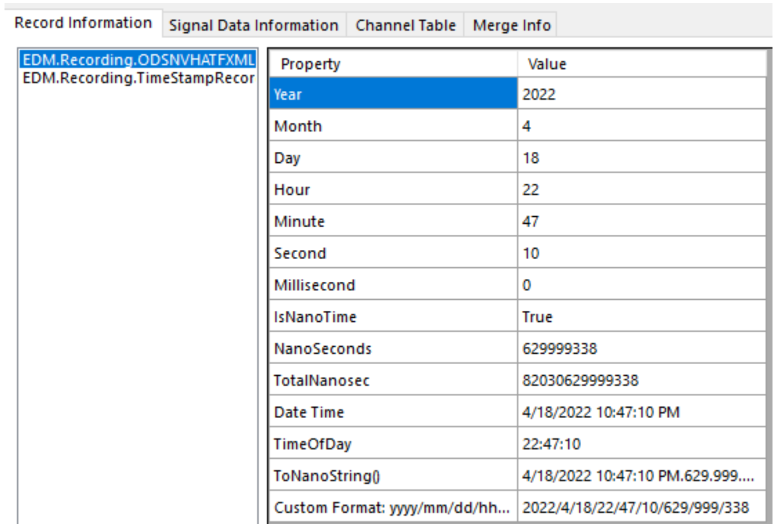
The below images show the various type of data stored in a ATFX file:

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
EDM.Recording.ODSNVHATFXML

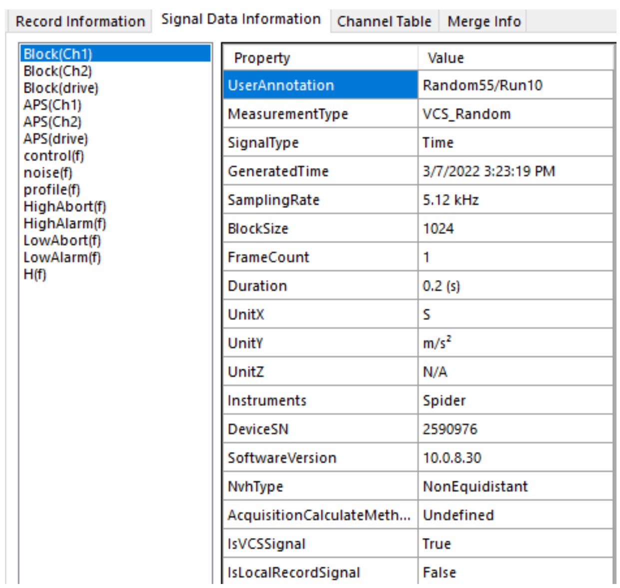
Property	Value
User	Admin
Instruments	Spider
TestNote	Random55/Run10
Name	SIG0010
RecordingPath	C:\Users\KevinCheng\Downloa...
Type	ODS_ATF_XML
RecordingTypeName	ASAM ODS Format - XML
Version	10.0.8.30
DeviceSNs	2590976
MasterSN	2590976
UserAnnotation	Random55/Run10
MeasurementType	VCS_Random
Time Zone	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 infromation regarding the recording create time and nanoseconds



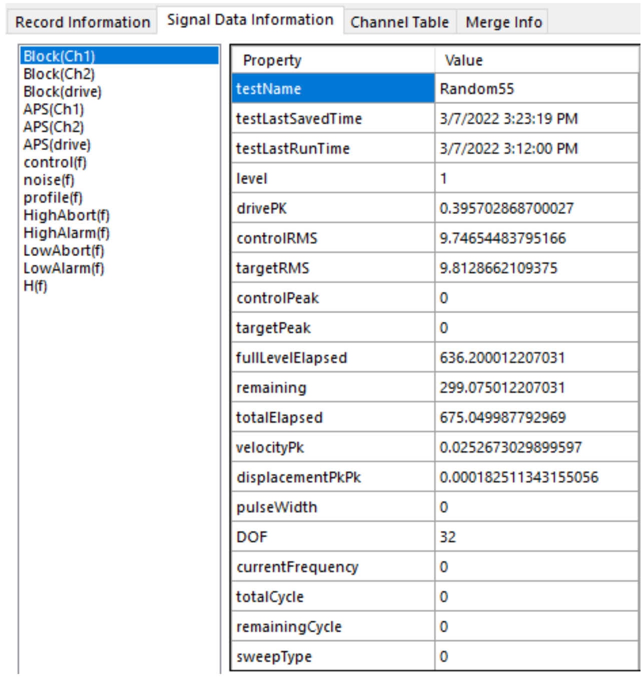
Property	Value
Year	2022
Month	4
Day	18
Hour	22
Minute	47
Second	10
Millisecond	0
IsNanoTime	True
NanoSeconds	629999338
TotalNanosec	82030629999338
Date Time	4/18/2022 10:47:10 PM
TimeOfDay	22:47:10
ToNanoString()	4/18/2022 10:47:10 PM.629.999....
Custom Format: yyyy/mm/dd/hh...	2022/4/18/22/47/10/629/999/338

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



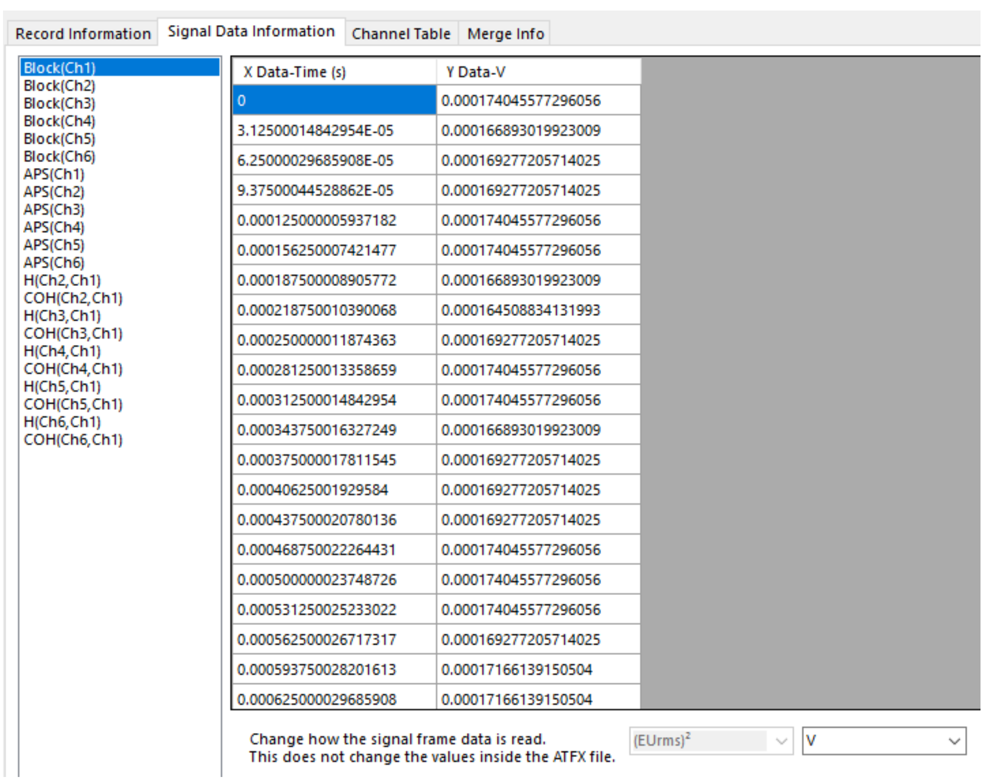
Property	Value
UserAnnotation	Random55/Run10
MeasurementType	VCS_Random
SignalType	Time
GeneratedTime	3/7/2022 3:23:19 PM
SamplingRate	5.12 kHz
BlockSize	1024
FrameCount	1
Duration	0.2 (s)
UnitX	S
UnitY	m/s^2
UnitZ	N/A
Instruments	Spider
DeviceSN	2590976
SoftwareVersion	10.0.8.30
NvhType	NonEquidistant
AcquisitionCalculateMeth...	Undefined
IsVCSSignal	True
IsLocalRecordSignal	False

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



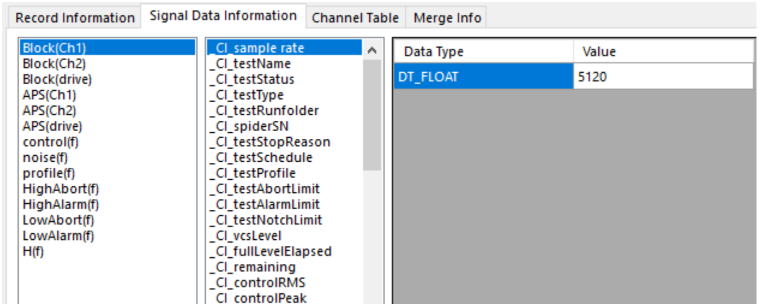
Property	Value
testName	Random55
testLastSavedTime	3/7/2022 3:23:19 PM
testLastRunTime	3/7/2022 3:12:00 PM
level	1
drivePK	0.395702868700027
controlRMS	9.74654483795166
targetRMS	9.8128662109375
controlPeak	0
targetPeak	0
fullLevelElapsed	636.200012207031
remaining	299.075012207031
totalElapsed	675.049987792969
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. There may be a chance that the data displayed in the ATFX API is different from what is displayed on EDM. This is due to the spectrum type being a display parameter and not saved in the ATFX file, thus it will default to $EUrms^2$.

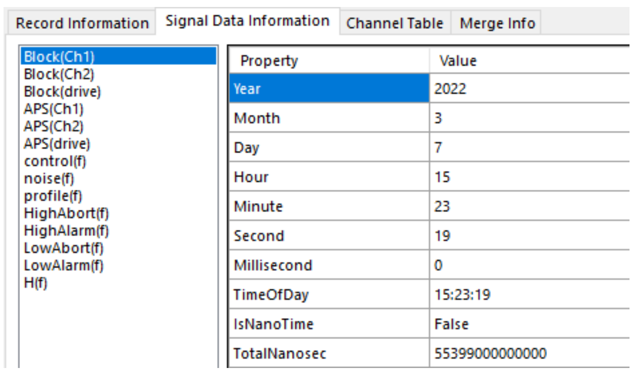


X Data-Time (s)	Y Data-V
0	0.000174045577296056
3.12500014842954E-05	0.000166893019923009
6.25000029685908E-05	0.000169277205714025
9.37500044528862E-05	0.000169277205714025
0.000125000005937182	0.000174045577296056
0.000156250007421477	0.000174045577296056
0.000187500008905772	0.000166893019923009
0.000218750010390068	0.000164508834131993
0.000250000011874363	0.000169277205714025
0.000281250013358659	0.000174045577296056
0.000312500014842954	0.000174045577296056
0.000343750016327249	0.000166893019923009
0.000375000017811545	0.000169277205714025
0.00040625001929584	0.000169277205714025
0.000437500020780136	0.000169277205714025
0.000468750022264431	0.000174045577296056
0.000500000023748726	0.000174045577296056
0.000531250025233022	0.000174045577296056
0.000562500026717317	0.000169277205714025
0.000593750028201613	0.00017166139150504
0.000625000029685908	0.00017166139150504

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

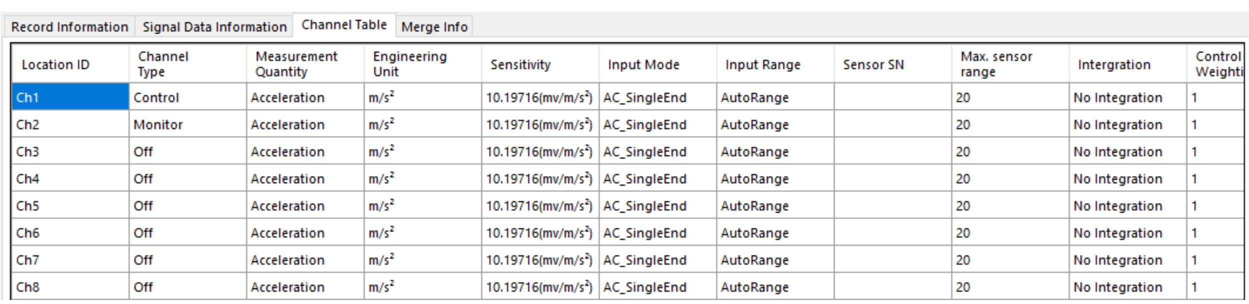


7) Signal Generate Time – Contains more advance information regarding a signal or atfx file generated time.



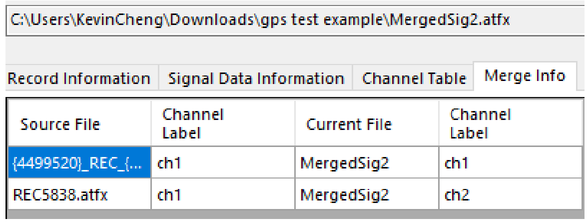
Property	Value
Year	2022
Month	3
Day	7
Hour	15
Minute	23
Second	19
Millisecond	0
TimeOfDay	15:23:19
IsNanoTime	False
TotalNanosec	55399000000000

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



Location ID	Channel Type	Measurement Quantity	Engineering Unit	Sensitivity	Input Mode	Input Range	Sensor SN	Max. sensor range	Intergration	Control Weighti
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 mutiple other atfx files if the file is merged.



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
```

There are 2 additional packages that the python demo scripts used to plot out the signal frame data and easily convert a C# array to a Python array, Matplotlib and Numpy.

```
pip install matplotlib

pip install numpy
```

If for some reason the install command returns a fatal error in launcher unable to create process using ‘ ” ” ’ then adding python -m to the pip install will work around the issue.

After installing the packages, 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.

```
#---Pythonnet clr import
import clr
# Change file path here to whereever the DLL files are
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')

import numpy as np
import matplotlib.pyplot as plt

#---C# .NET imports & dll imports
from EDM.Recording import *
from EDM.RecordingInterface import *
from ASAM.ODS.NVH import *
```

```
from EDM.Utils import *
from Common import *
from Common import _SpectrumScalingType
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 methods 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 ShowGPSInfo(recordingPath):
    recording = ODSNVHATFXMLRecording(recordingPath)

    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(nvhEnvironment.TimeZoneString):
            print("Time Zone: ", nvhEnvironment.TimeZoneString)

        print("Created Time (Local): ", nvhRec.RecordingProperty.CreateTime)
        print("Created Time (UTC): ",
Utils.GetUTCTime(nvhRec.RecordingProperty.CreateTime, None))

#---Main Code
print("Running Main Code")

# Change file path here to whereever signal or recording files are
recordingPath = "C:\\Users\\KevinCheng\\Downloads\\gps test example\\"
# ATFX file path, change contain the file name and correctly reference it in
RecordingManager.Manager.OpenRecording
recordingPathTS = recordingPath + "{4499520}_REC_{20220419}(1).atfx"

#OpenRecording(string, out IRecording)
# dummy data is required for the OpenRecording for it to correctly output data
# Make sure to reference the correct file string
dummyTest1, recording = RecordingManager.Manager.OpenRecording(recordingPathTS, None)

print("\nRecording Properties\n")
for prop in Utility.GetListOfProperties(recording.RecordingProperty):
```

```
    print(prop[0], prop[1])

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

print("\nSignal 1 Properties\n")
for prop in Utility.GetListOfProperties(recording.Signals[0].Properties):
    print(prop[0], prop[1])

print("\nSignal 1 Properties GeneratedTime\n")
for prop in Utility.GetListOfProperties(recording.Signals[0].Properties.GeneratedTime):
    print(prop[0], prop[1])
```

Example Print Statements

```
Running Main Code


Recording Properties


User Unknown Owner
Instruments GRS
TestNote Untitled Test Note
RecordingName {4499520}_REC_{20220419}(1)
RecordingPath C:\Users\KevinCheng\Downloads\gps test example\{4499520}_REC_{20220419}(1).atfx
RecordingType ODS_ATF_XML
RecordingTypeName ASAM ODS Format - XML
SavingVersion 10.0.8.34
DeviceSNs 4499520
MasterSN 4499520
MeasurementType None


Recording GPS Properties


GPS Enabled:  True
Longitude:  0.0
Latitude:  37.38046
Altitude:  12.42
Nanoseconds Elapsed:  629999338
```

Time Zone: Eastern Standard Time;-300;(UTC-05:00) Eastern Time (US & Canada);Eastern Standard Time;Eastern Daylight Time;[01:01:0001;12:31:2006;60;[0;02:00:00;4;1;0;];[0;02:00:00;10;5;0;];][01:01:2007;12:31:9999;60;[0;02:00:00;3;2;0;];[0;02:00:00;11;1;0;];];

Created Time (Local): 4/18/2022 2:47:10 PM

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

Signal 1 Properties

MeasurementType None

SignalType Time

GeneratedTime 4/18/2022 2:47:10 PM.629.999.338

SamplingRate 51.20 kHz

BlockSize 1793024

FrameCount 1

Duration 35.02 (s)

UnitX Time (s)

UnitY V

UnitZ N/A

Instruments GRS

DeviceSN 4499520

SoftwareVersion 10.0.8.34

NvhType Equidistant

AcquisitionCalculateMethod Undefined

IsVCSSignal False

IsLocalRecordSignal False

Signal 1 Properties GeneratedTime

Year 2022

Month 4

Day 18

Hour 14

Minute 47 Second 10 Millisecond 0 TimeOfDay 14:47:10 IsNanoTime True TotalNanoSeconds 53230629999338

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 32-bit** version.

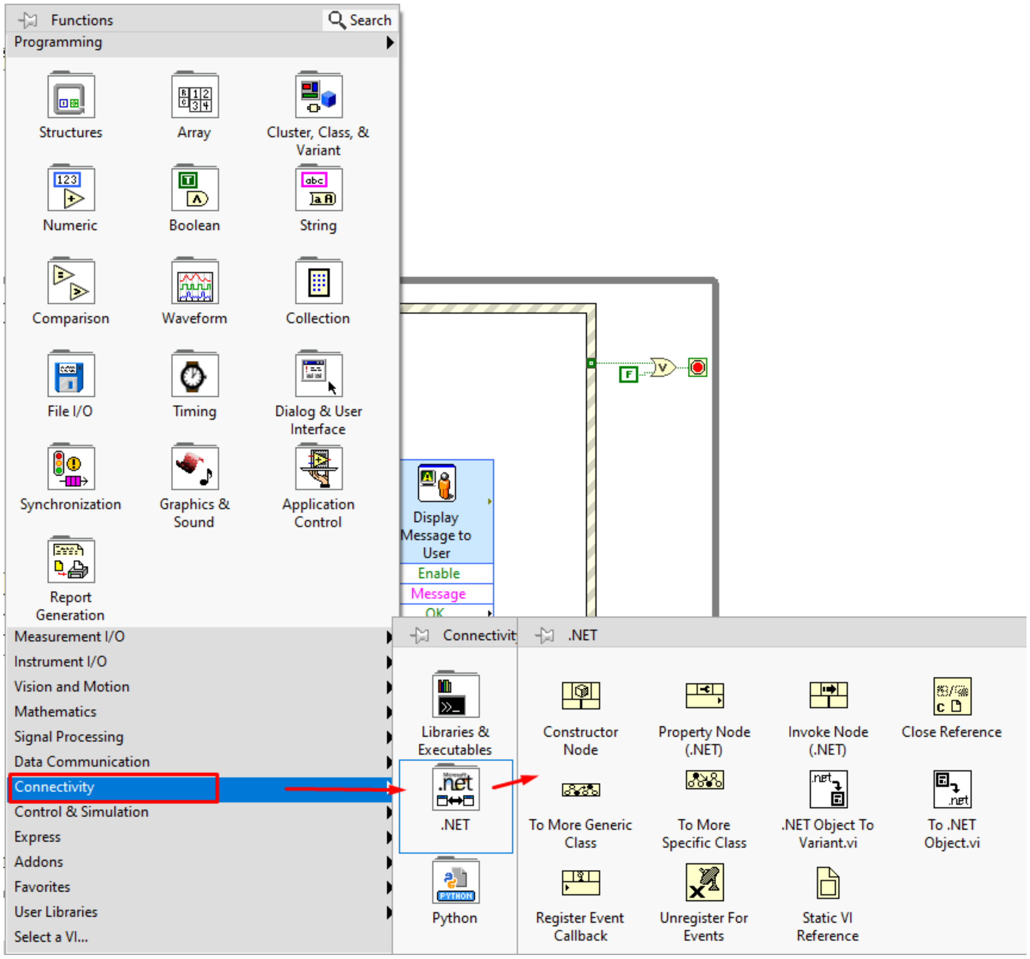
Importing C# DLL files

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

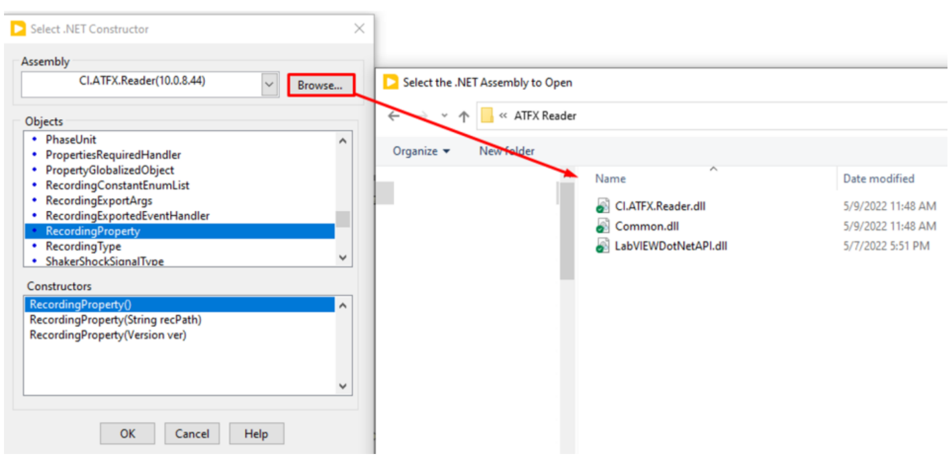
https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YGggCAG&l=en-US

The ATFX API for LabVIEW comes with an additional DLL file called **LabVIEWDotNetAPI** that provides methods and properties to open and read ATFX files in LabVIEW. It is similar to the C# demo code except encapsulated into a library. Thus if there are additional methods or properties needed, the customer must send a request to Crystal Instrument software team.

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.

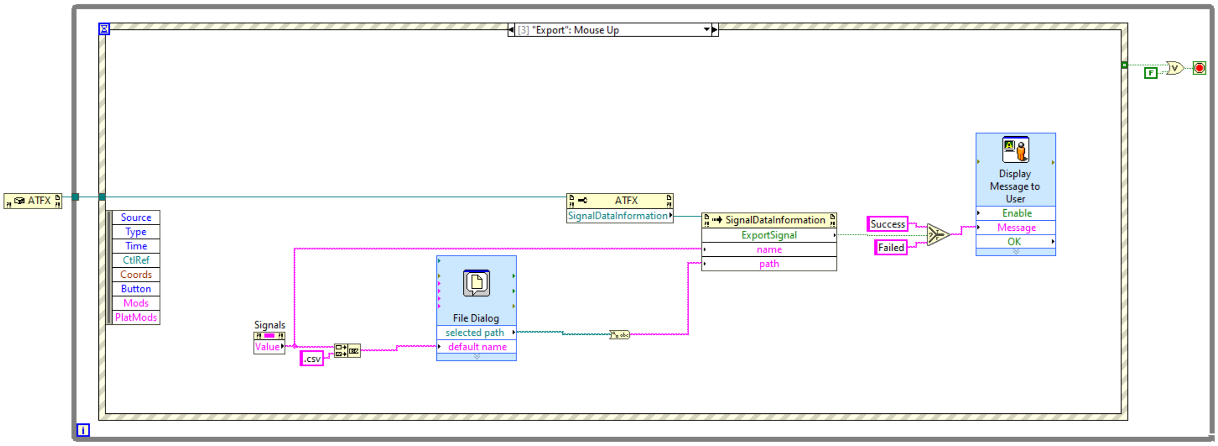


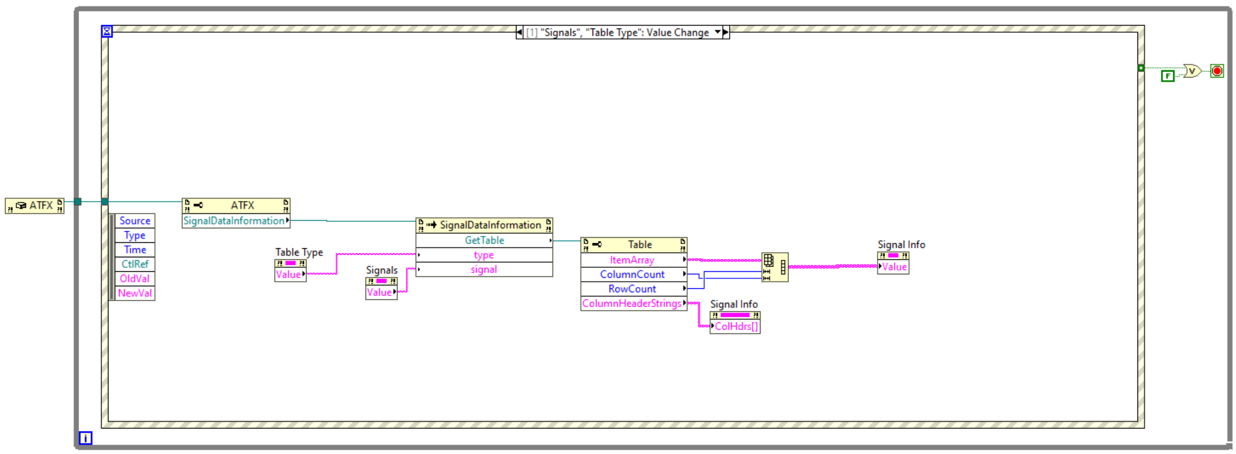
If the user selects the **Constructor Node** and place into the diagram, another window will pop up for selecting the .NET constructor reference. If the ATFX API dll files are not in the assembly list, then users can click **Browse** and add in the dll files.

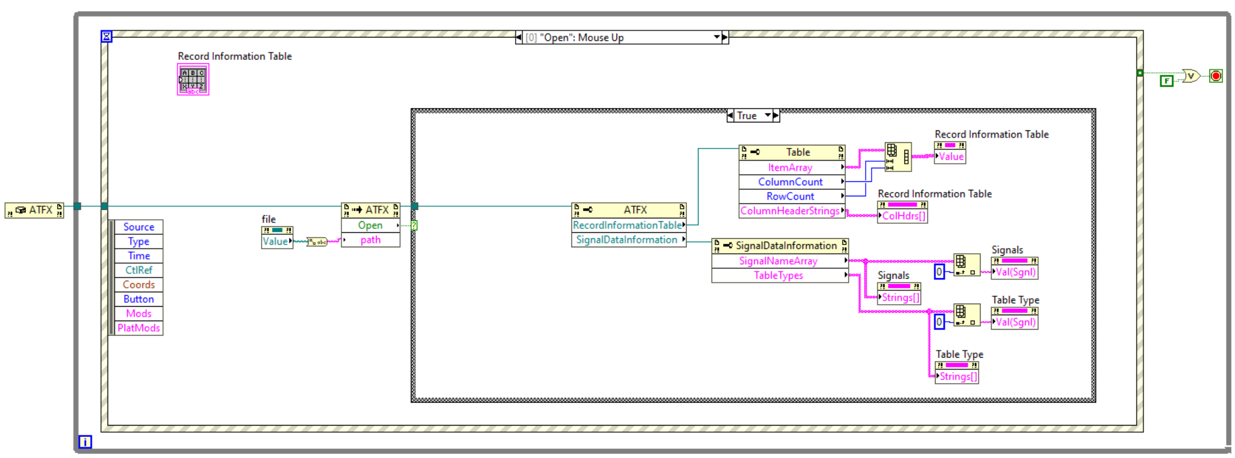


LabVIEW Block Diagram Example

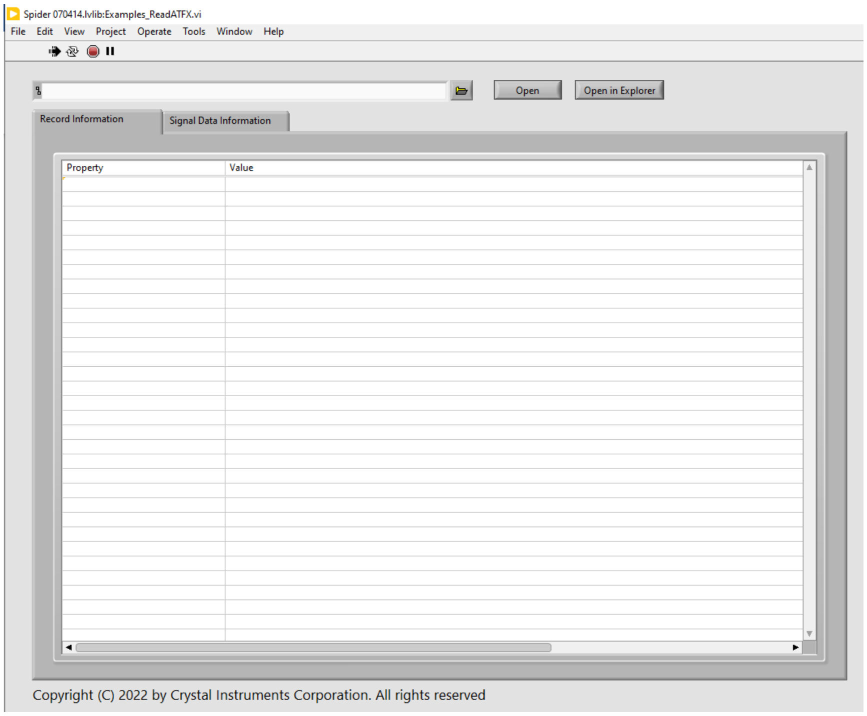
The following shows the block diagram used to open the ATFX file and display its data from the **Examples_ReadATFX.vi** file.



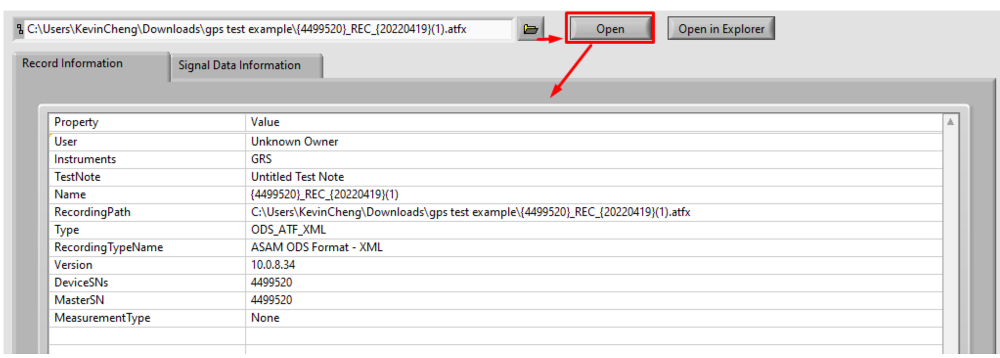




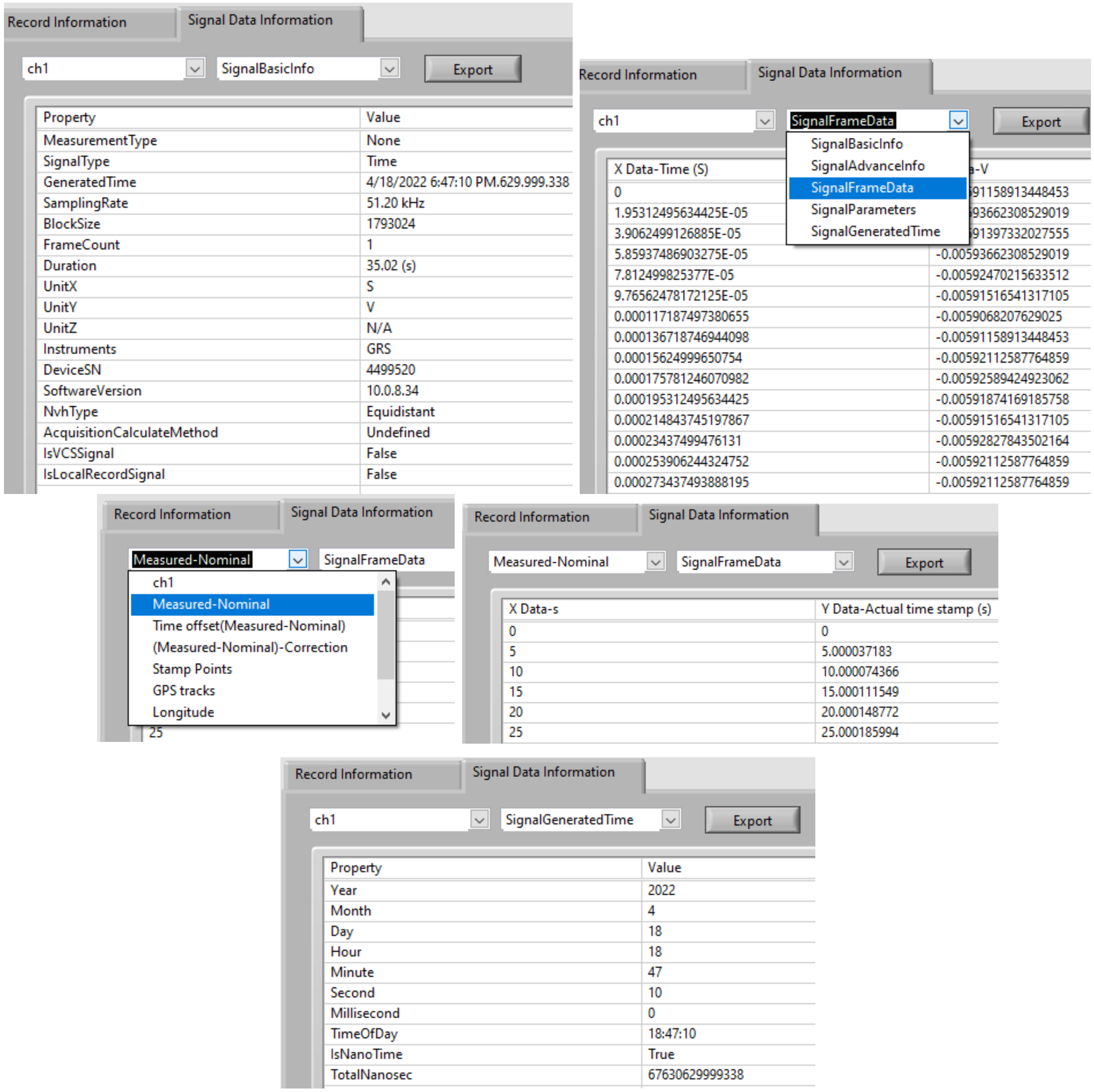
The following shows the GUI of the ATFX API LabView Reader and its usage.



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**().

```
% Load common and reader dll
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 methods and properties similar to C#.

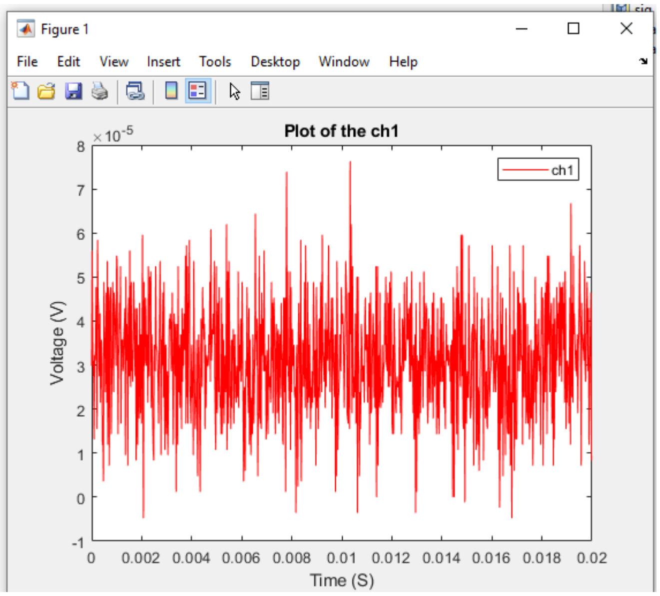
An example below shows how to open a recording and display its recording properties and signal frame data.

```
% Create a atfx recording instance
rec =
EDM.Recording.ODSNVHATFXMLRecording('C:\Users\KevinCheng\Documents\EDM\test\Random6
9\Run3 Jul 01, 2022 11-20-16\SIG0004.atfx');

% Use item function to get a time 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.TimeZoneString));
disp(System.String.Format("Created Time
(Local):{0}",rec.RecordingProperty.CreateTime));
disp(System.String.Format("Created Time
(UTC):{0}",Common.Utils.GetUTCTime(rec.RecordingProperty.CreateTime, [])));
disp(System.String.Format("Nanoseconds
Elapsed:{0}",rec.Measurement.NanoSecondElapsed));

dateTimeNano = Common.DateTimeNano(rec.RecordingProperty.CreateTime,
rec.Measurement.NanoSecondElapsed);
disp(System.String.Format("DateTimeNano Object:{0}",dateTimeNano));

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

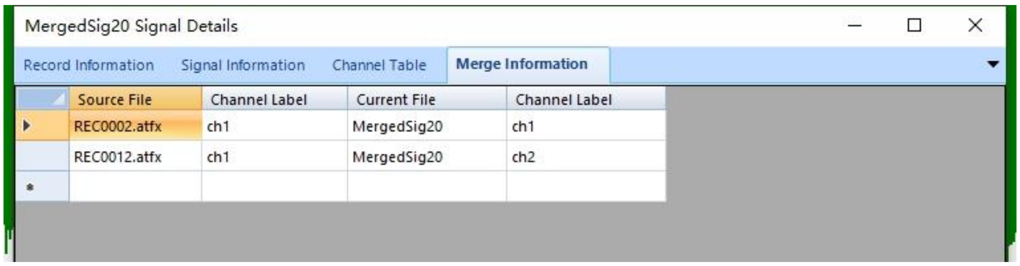
```
Name:ch1

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
>>
```

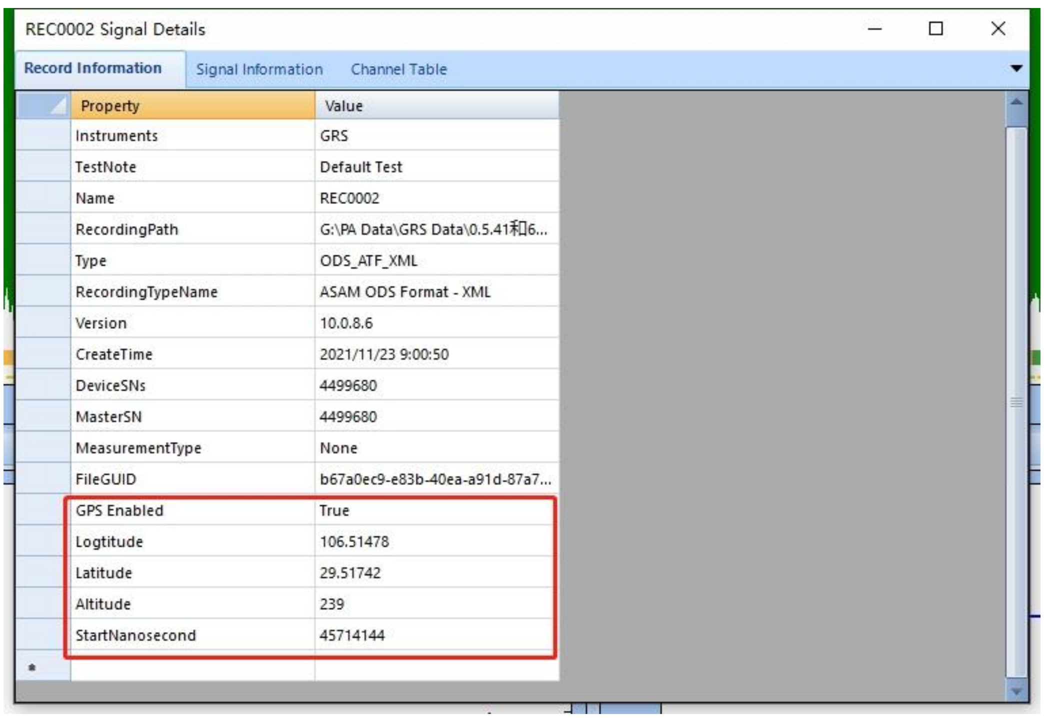


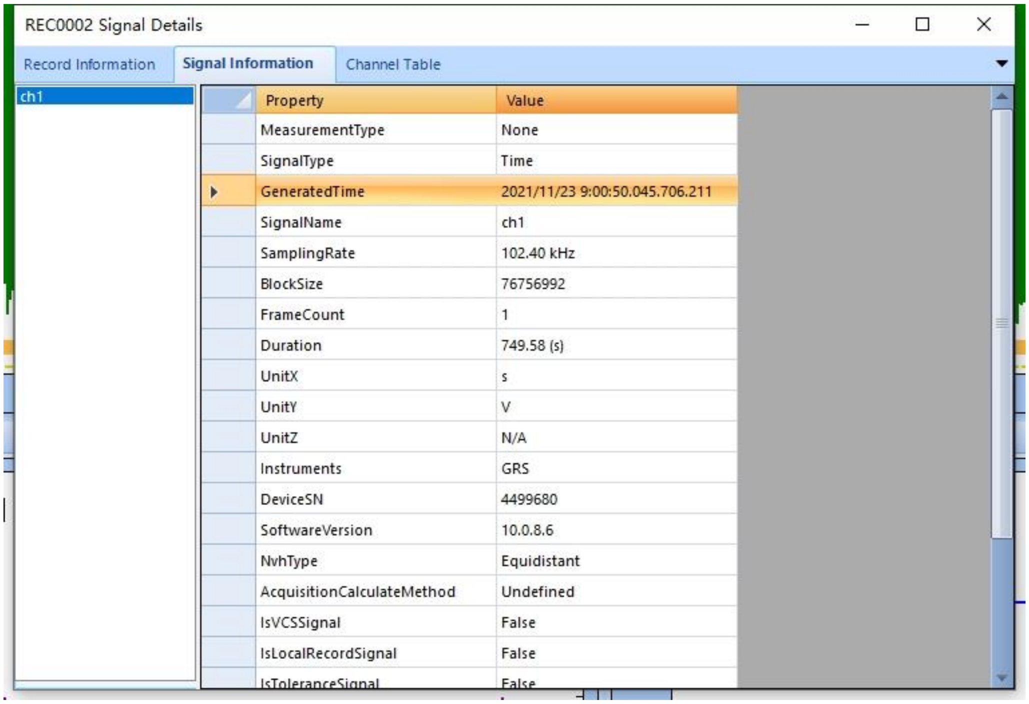
Post Analysis Software Integrates ATFX API

The Feature that Utilizes ATFX Reader API in PA Software

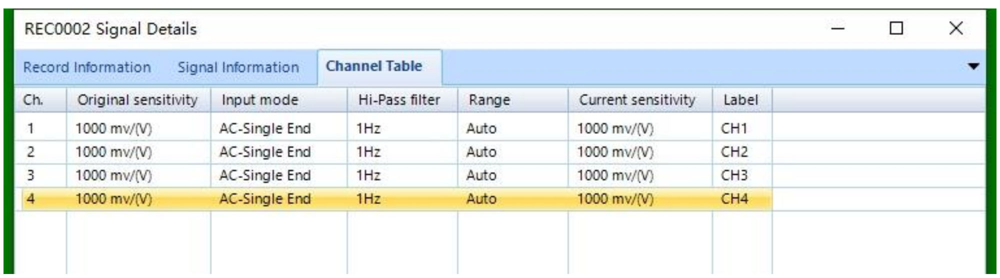
The following screenshots of the Post Analysis Software shows a feature that integrates ATFX Reader API, which reads and shows all the information in atfx files that are created by Crystal Instruments products. The ATFX Reader API not only can be integrated in software products of Crystal Instruments,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







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
IsVCSSignal	False
IsLocalRecordSignal	False
IsToleranceSignal	False



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

Appendix

Time Domain Signals

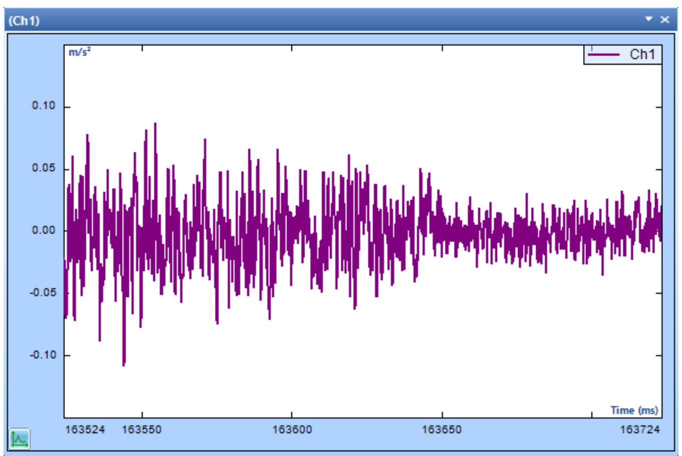
Time domain signals displays signal amplitude (y-axis) over a period of time (x-axis). These types of signals are not affected by changes in spectrum types.

Time Stream

The time stream signals are the raw time waveforms applied to the input channels. They are displayed with relative time on the Y-axis.

They are a live feed of time data, useful for live monitoring a signal in the time domain. Thus, Time Stream signals are not saved into the ATFX file.

A Time Stream signal from an EDM VCS Random test:



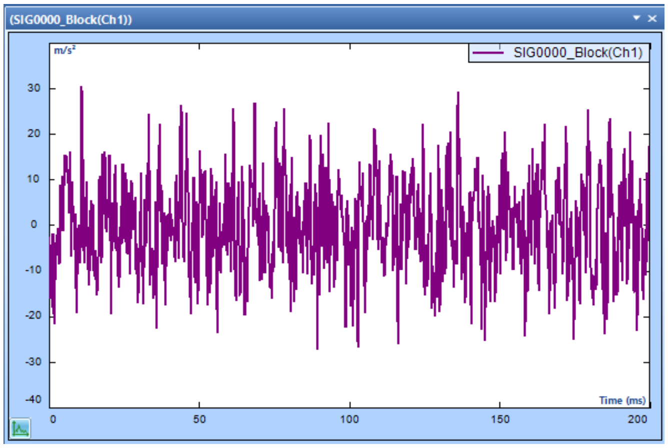
(Ch1)

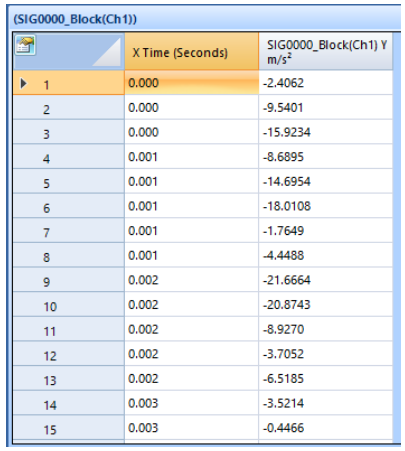
	X Time (Seconds)	Ch1 Y m/s²
1	163.524	-0.0213
2	163.524	-0.0236
3	163.524	-0.0664
4	163.524	-0.0356
5	163.525	-0.0698
6	163.525	-0.0626
7	163.525	-0.0303
8	163.525	-0.0300
9	163.525	0.0350
10	163.526	0.0385
11	163.526	0.0104
12	163.526	-0.0127
13	163.526	-0.0250
14	163.526	0.0029
15	163.526	0.0268

Time Block

Time Blocks are a contiguous segment of time domain data, which can then be transformed into the frequency domain. The block size is often a power of two.

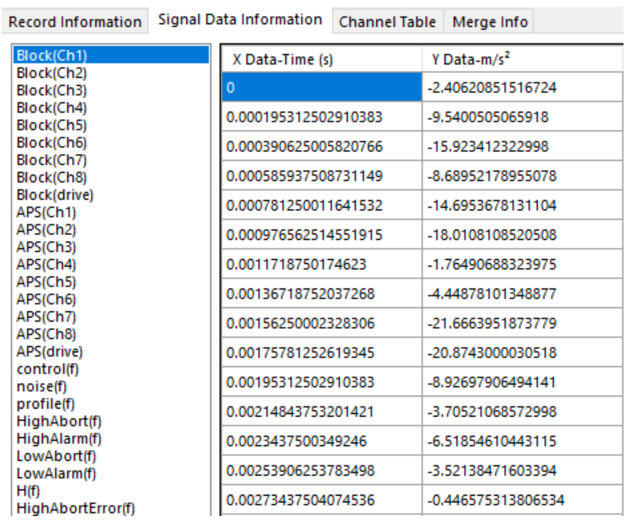
A Time Block signal from an EDM VCS Random test:




(SIG0000_Block(Ch1))

	X Time (Seconds)	SIG0000_Block(Ch1) Y m/s²
1	0.000	-2.4062
2	0.000	-9.5401
3	0.000	-15.9234
4	0.001	-8.6895
5	0.001	-14.6954
6	0.001	-18.0108
7	0.001	-1.7649
8	0.001	-4.4488
9	0.002	-21.6664
10	0.002	-20.8743
11	0.002	-8.9270
12	0.002	-3.7052
13	0.002	-6.5185
14	0.003	-3.5214
15	0.003	-0.4466

ATFX API C# Demo display



X Data-Time (s)	Y Data-m/s²
0	-2.40620851516724
0.000195312502910383	-9.5400505065918
0.000390625005820766	-15.923412322998
0.000585937508731149	-8.68952178955078
0.000781250011641532	-14.6953678131104
0.000976562514551915	-18.0108108520508
0.0011718750174623	-1.76490688323975
0.00136718752037268	-4.44878101348877
0.00156250002328306	-21.6663951873779
0.00175781252619345	-20.8743000030518
0.00195312502910383	-8.92697906494141
0.00214843753201421	-3.70521068572998
0.0023437500349246	-6.51854610443115
0.00253906253783498	-3.52138471603394
0.00273437504074536	-0.446575313806534

Frequency Domain Signals

Frequency domain signals displays signal amplitude (y-axis) over a frequency range (x-axis). Frequency domain signals are usually expressed in Hz and calculated from an equivalent "block" of time domain data (also known as "frame") through mathematical transforms, such as the Fourier Transform.

Here is a list of frequency signals and their short form:

Frequency Spectrum Full Name	EDM / ATFX Spectrum Abbreviation
Auto Power Spectrum	APS
Frequency Response Function	FRF H
Fast Fourier Transform	FFT

Cross Power Spectrum	CPS
Coherence Function	COH
Sine	Spectrum
Shock Response Spectrum	MaxiSRS PosSRS NegSRS
Order	ORDSpec
Octave	OCT

Fast Fourier Transform Spectral Analysis Linear (FFT)

Digital signal processing technology includes FFT based frequency analysis, digital filters and many other topics. This chapter introduces the FFT based frequency analysis methods that are widely used in all dynamic signal analyzers. CoCo has fully utilized the FFT frequency analysis methods and various real time digital filters to analyze the measurement signals.

The Fourier Transform is a transform used to convert quantities from the time domain to the frequency domain and vice versa, usually derived from the Fourier integral of a periodic function when the period grows without limit, often expressed as a Fourier transform pair. In the classical sense, a Fourier transform takes the form of:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}\,dt$$

where:

x(t) - continuous time waveform

f - frequency variable

j - complex number

X(f) - Fourier transform of x(t)

Mathematically the Fourier Transform is defined for all frequencies from negative to positive infinity. However, the spectrum is usually symmetric and it is common to only consider the single-sided spectrum which is the spectrum from zero to positive infinity. For discrete sampled signals, this can be expressed as:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}$$

where:

x(n) - samples of time waveform

n - running sample index

N - total number of samples or "frame size"

k - finite analysis frequency, corresponding to "FFT bin centers"

X(k) - discrete Fourier transform of *x(k)*

In most DSA products, a Radix-2 DIF FFT algorithm is used, which requires that the total number of samples must be a power of 2 (total number of samples in FFT = 2^m, where m is an integer).

Selecting different spectrum types will not affect the FFT spectrum in Real + Imaginary values.

Linear Spectrum

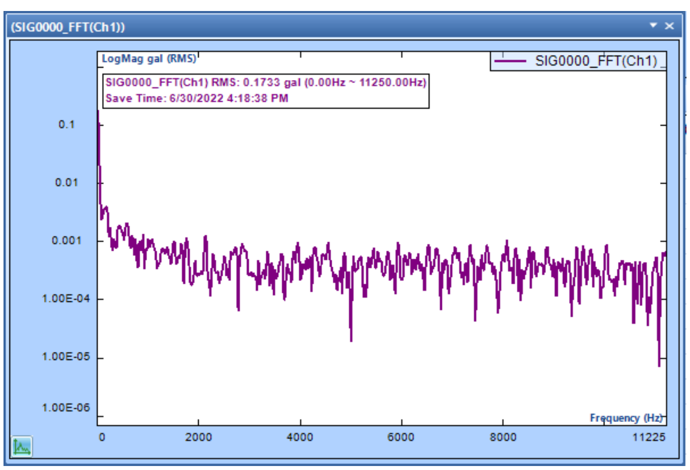
A linear spectrum is the Fourier transform of windowed time domain data. The linear spectrum is useful for analyzing periodic signals. You can extract the harmonic amplitude by reading the amplitude values at those harmonic frequencies.

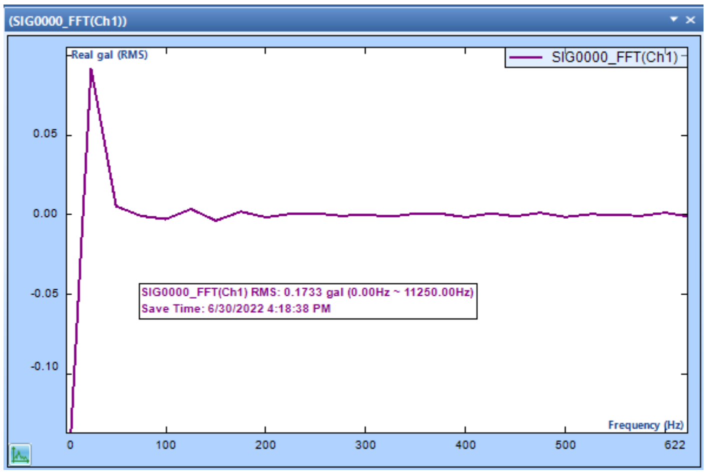
An averaging technique is often used when synchronized triggering is applied. Because the averaging is taking place in the linear spectrum domain, or equivalently, in the time domain, based on the principles of linear transform, averaging makes no sense unless a synchronized trigger is used.

In many DSA products, amplitude correction is automatically applied when selecting different Spectrum Types.

The linear spectrum is saved internally in the complex data format with real and imaginary parts. Therefore, you should be able to view the real, imaginary, amplitude, or the phase part of the spectrum.

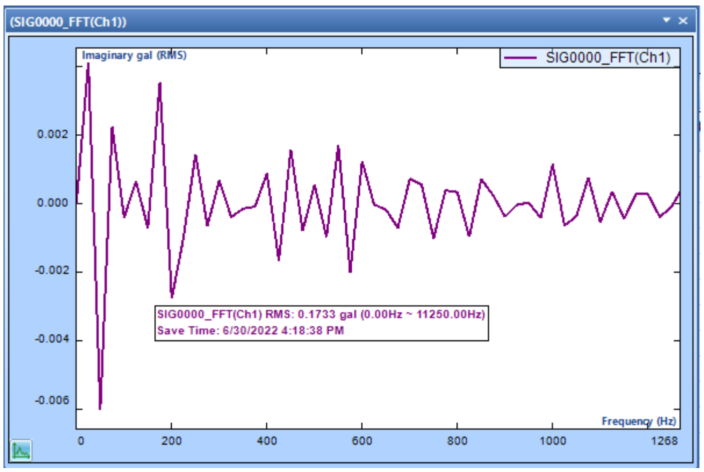
An FFT signal from an EDM DSA FFT Analysis test:





(FFT(Ch1))

	X Frequency (Hz)	FFT(Ch1) Y-Re Real gal (RMS)
1	0.00	-0.1908
2	25.00	0.0917
3	50.00	0.0055
4	75.00	-0.0007
5	100.00	-0.0026
6	125.00	0.0036
7	150.00	-0.0036
8	175.00	0.0020
9	200.00	-0.0015
10	225.00	0.0007
11	250.00	0.0010
12	275.00	-0.0006
13	300.00	0.0001
14	325.00	-0.0011
15	350.00	0.0008



(FFT(Ch1))

	X Frequency (Hz)	FFT(Ch1) Y-Re Imaginary gal (RMS)
1	0.00	0.0000
2	25.00	0.0041
3	50.00	-0.0060
4	75.00	0.0023
5	100.00	-0.0004
6	125.00	0.0007
7	150.00	-0.0007
8	175.00	0.0035
9	200.00	-0.0028
10	225.00	-0.0010
11	250.00	0.0014
12	275.00	-0.0007
13	300.00	0.0007
14	325.00	-0.0004
15	350.00	-0.0001

ATFX API C# Demo display

The ATFX API will read the FFT in Real & Imaginary values.

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

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)
CPS(Ch2,Ch1)
CPS(Ch3,Ch1)
CPS(Ch4,Ch1)
CPS(Ch5,Ch1)
H(Ch2,Ch1)
COH(Ch2,Ch1)
H(Ch3,Ch1)
COH(Ch3,Ch1)
H(Ch4,Ch1)
COH(Ch4,Ch1)
H(Ch5,Ch1)
COH(Ch5,Ch1)
FFT(Ch1)
FFT(Ch2)
FFT(Ch3)
FFT(Ch4)
FFT(Ch5)

X Data-Frequency (Hz)	Y Data-Real gal (RMS)	Y data-Imaginary gal (RMS)
0	-0.190758779644966	0
25	0.0916995480656624	0.00413563661277294
50	0.00550242513418198	-0.0060243490152061
75	-0.000699079886544496	0.00226424890570343
100	-0.00262495945207775	-0.000412846391554922
125	0.00364094506949186	0.000653044378850609
150	-0.00360224535688758	-0.000720723997801542
175	0.00202021608129144	0.0035436199977994
200	-0.00151090265717357	-0.00277522136457264
225	0.00065707485191524	-0.000979538075625896
250	0.00103858741931617	0.00144634069874883
275	-0.000587686372455209	-0.000658069388009608
300	0.000149876897921786	0.000677179836202413
325	-0.00107129942625761	-0.000395542243495584
350	0.000813271617516875	-0.000147657367051579

Auto Power Spectrum (APS)

Spectral analysis of data has for a long time been popular in characterizing the operation of mechanical and electrical systems. A type of spectral analysis, the power spectrum (and power spectral density), is especially popular because a "power" measurement in the frequency domain is one that engineers readily accept and apply in their solutions to problems. Single channel measurements (auto-power spectra) and two channel measurements (cross-power spectra) have both played important roles.

In many DSA products, Power Spectrum Analysis is a general name for computing the following three spectrum types:

- Power Spectrum: The unit is EU^2
- Power Spectrum Density(PSD): The unit is EU^2/Hz
- Energy Spectrum Density(ESD): The unit is EU^2S/Hz

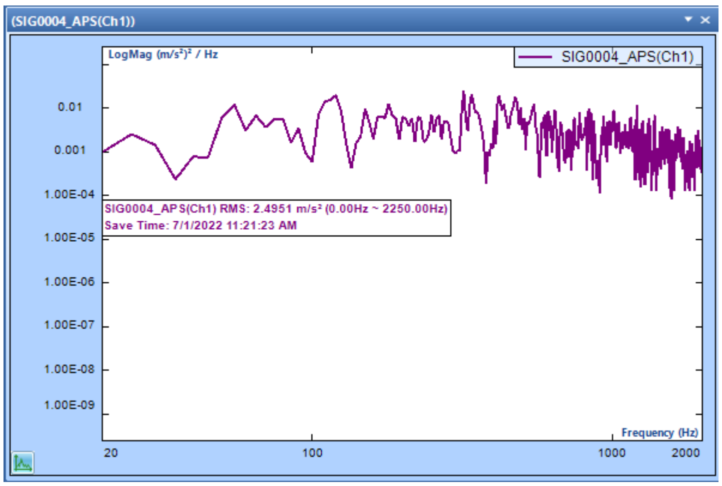
In power spectrum measurements, window amplitude correction is used to get un-biased final spectrum readings at specific frequency. In PSD or ESD Spectrum measurements, window energy correction is always used to get an un-biased spectral density reading.

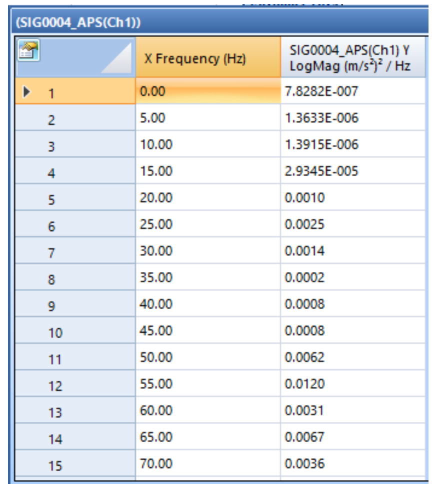
The magnitude of the frequency components of signals are collectively called the amplitude spectrum. In many applications, the quantity of interest is the power or the rate of energy transfer proportional to the squared magnitude of the frequency components. The average squared magnitudes of all the DFT frequency lines are collectively referred to as the Power Spectrum, G_{xx}.

The averaging process is more properly termed an ensemble average, wherein the squared amplitude from N signal blocks at each measured frequency, f, are averaged together. Letting an asterisk (*) denote conjugation of a complex number, the "power" averaging process is defined by:

$$\boldsymbol{G_{xx}(f) = |X(f)|^2 = \frac{1}{N}\sum_{k=1}^{N} X_k(f)X_k^*(f)}$$

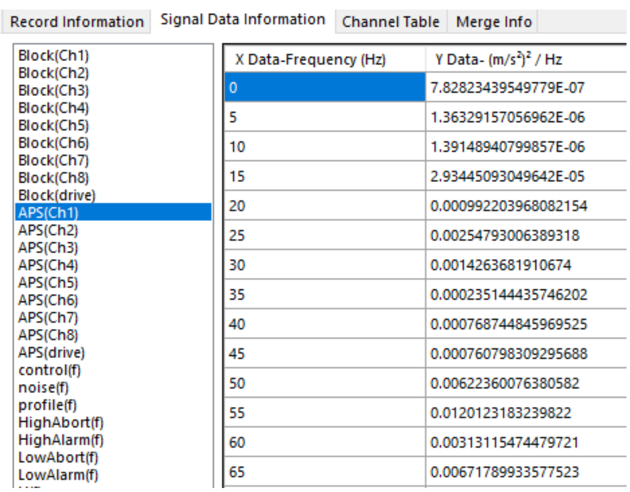
APS signals from an EDM VCS Random test:




(SIG0004_APS(Ch1))

	X Frequency (Hz)	SIG0004_APS(Ch1) Y LogMag $(m/s^2)^2$ / Hz
1	0.00	7.8282E-007
2	5.00	1.3633E-006
3	10.00	1.3915E-006
4	15.00	2.9345E-005
5	20.00	0.0010
6	25.00	0.0025
7	30.00	0.0014
8	35.00	0.0002
9	40.00	0.0008
10	45.00	0.0008
11	50.00	0.0062
12	55.00	0.0120
13	60.00	0.0031
14	65.00	0.0067
15	70.00	0.0036

ATFX API C# Demo display


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

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
Block(Ch6)
Block(Ch7)
Block(Ch8)
Block(drive)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)
APS(Ch6)
APS(Ch7)
APS(Ch8)
APS(drive)
control(f)
noise(f)
profile(f)
HighAbort(f)
HighAlarm(f)
LowAbort(f)
LowAlarm(f)

X Data-Frequency (Hz)	Y Data- $(m/s^2)^2$ / Hz
0	7.82823439549779E-07
5	1.36329157056962E-06
10	1.39148940799857E-06
15	2.93445093049642E-05
20	0.000992203968082154
25	0.00254793006389318
30	0.0014263681910674
35	0.000235144435746202
40	0.000768744845969525
45	0.000760798309295688
50	0.00622360076380582
55	0.0120123183239822
60	0.00313115474479721
65	0.00671789933577523

Spectrum Types

Several Spectrum Types are given for both Linear Spectrum and Power Spectrum measurements in CoCo and EDM. The concept of spectrum type is explained below in detail.

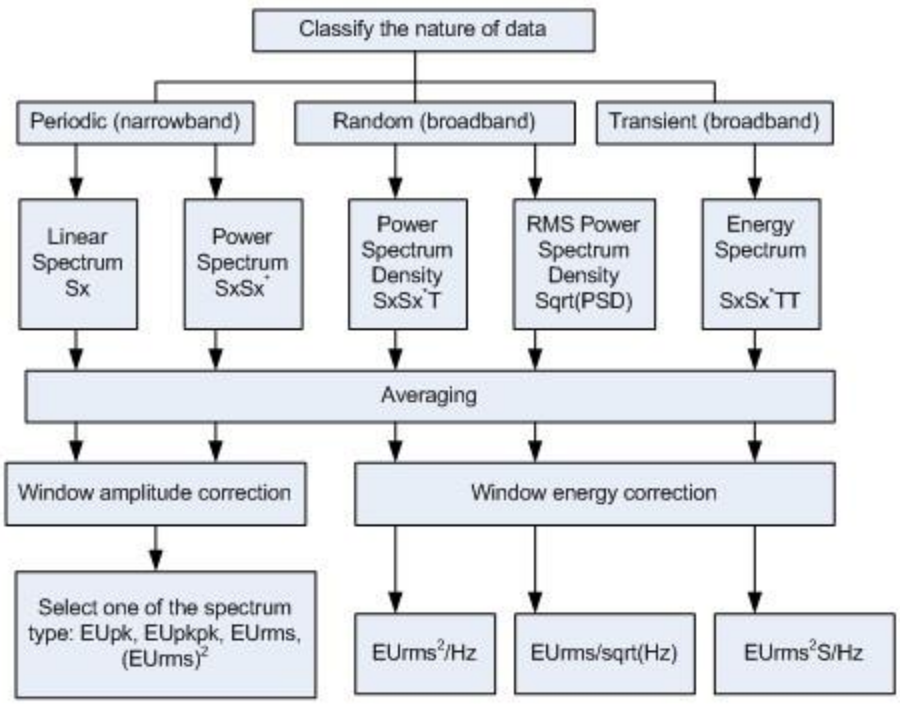
First let’s consider the signals with periodic nature. These can be the signals measured from a rotating machine, bearing, gearing, or anything that repeats. In this case we would be interested in amplitude changes at fundamental frequencies, harmonics or sub-harmonics. In this case, you can choose a spectrum type of EU_{pk}, EU_{pkpk} or EU_{rms}.

A second scenario might consist of a signal with a random nature that is not necessarily periodic. It does not have obvious periodicity therefore the frequency analysis could not determine the “amplitude” at certain frequencies. However, it is possible to measure the r.m.s. level, or power level, or power density level over certain frequency bands for such random signals. In this case, you must select one of the spectrum types of EU_{rms}^2/Hz, or EU_{rms}/sqrt(Hz), which is called power spectral density, or root-mean squared density.

A third scenario might consist of a transient signal. It is neither periodic, nor stably random. In this case, must select a spectrum type as EU^2S/Hz, which is called energy spectrum.

In many applications, the nature of the data cannot be easily classified. Care must be taken to interpret the data when different spectrum types are used. For example, in the environmental vibration simulation, a typical test uses multiple sine tones on top of random profile, which is called Sine-on-Random. In this type application, you have to observe the random portion of the data in the spectrum with $EUrms^2/Hz$ and the sine portion of the data with EU_{pk}.

The image below shows a general flow-chart to choose one of the measurement techniques and spectrum types for linear or auto spectrum:



Flow chart to determine measurement technique for various signal types.

The following figures illustrate the results of different measurement techniques on a 1 volt pure sine tone. The figures include RMS, Peak or Peak-Peak value for the amplitude, or power value corresponding to its amplitude.

Notice these readings can only be applied to a periodic signal. If you applied these measurement techniques to a signal with random nature, the spectrum would not be a meaningful representation of the signal.

It should also be noted that since a window is applied in time domain, which corresponds a convolution in the linear spectrum, we cannot have both a valid amplitude and correct energy correction at the same time. Use the flow chart to select appropriate spectrum types.

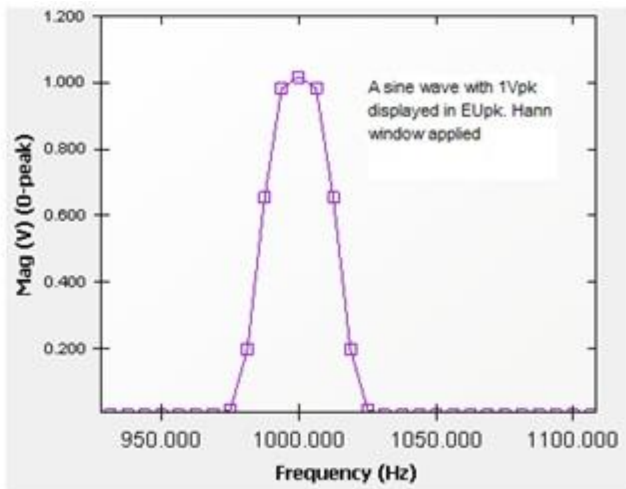
In a Linear Spectrum measurement, a signal is saved in its complex data format which includes both real and imaginary data. Then is averaging operation applied to the linear spectrum.

In a Power Spectrum measurement, the averaging operation is applied to the squared spectrum, which has only real part. Because of different averaging techniques, the final results of Linear Spectrum and Power Spectrum will be different even though the same spectrum type is used.

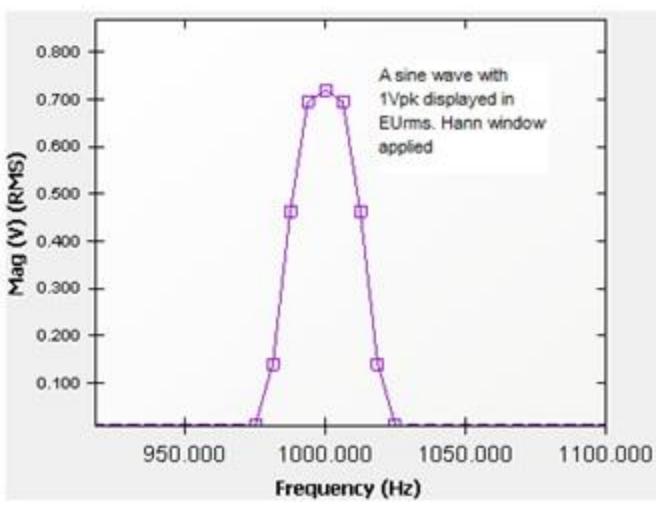
Spectrum Types selection only applies to Power Spectrum and Linear Spectrum signals. Spectrum Types do not apply to transfer functions, phase functions or coherence functions.

EU_{pk} or EU_{pkpk}

The EU_{pk} and EU_{pkpk} displays the peak value or peak-peak value of a periodic frequency component at a discrete frequency. These two spectrum types are suitable for narrowband signals.



A sine wave is measured with EUpk spectrum unit. The sine waveform has a 1V amplitude.



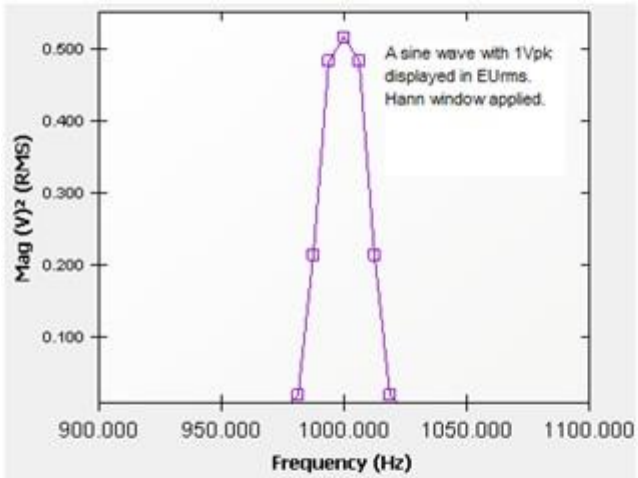
A sine wave is measured with EUrms spectrum unit. The peak reading is 0.707V. The sine waveform has a 1V amplitude.

EU_{rms}

The EU_{rms} displays the RMS value of a periodic frequency component at a discrete frequency. This spectrum type is suitable for narrowband signals.

$EU_{rms})^2$ Power spectrum

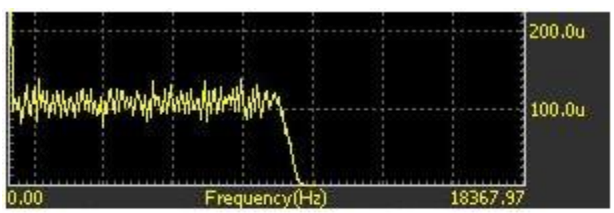
The $(EU_{rms})^2$ displays the power reading of a periodic frequency component at a discrete frequency. This spectrum type is suitable for narrowband signals.



A sine wave is measured with (EUrms)2 spectrum unit. The peak reading is 0.5V2. The sine waveform has a 1V amplitude.

EU_2/Hz, Power Spectrum Density

The EU_2/Hz is the spectrum unit used in power spectrum density (PSD) calculations. The unit is in engineering units squared divided by the equivalent filter bandwidth. This provides power normalized to a 1Hz bandwidth. This is useful for wideband, continuous signals. EU_2/Hz really should be written as $(EU_{rms})_2$/Hz. But probably due to the limitation of space, people put it as EU_2/Hz.



White noise with 1 volt RMS amplitude displays as 100 u Vrms2/Hz.

The image above shows a white noise signal with $1V_{rms}$ amplitude or $1V^2$ in power level. The bandwidth of the signal is approximately 10000 Hz and the V^2/Hz reading of the signal is around 0.0001 V^2/Hz. The 1 V RMS can be calculated as follows:

$1\ V_{rms} = \text{sqrt}\ (10000\text{Hz} * 0.0001\ V^2/\text{Hz})$

EU^2S/Hz, Energy Spectrum Density

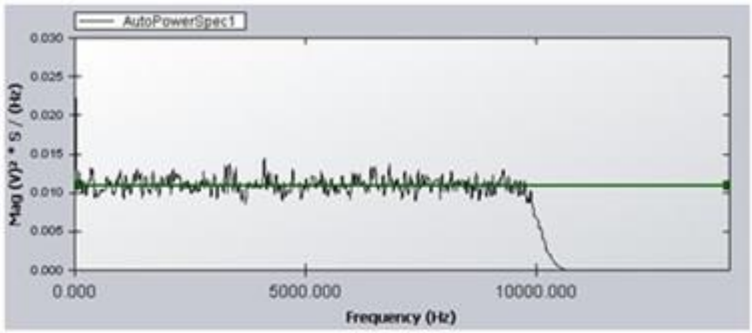
The EU^2S/Hz displays the signal in engineering units squared divided by the equivalent filter bandwidth, multiplied by the time duration of signal. This spectrum type provides energy normalized to a 1Hz bandwidth, or energy spectral density (ESD). It is useful for any signals when the purpose is to measure the total energy in the data frame.

The ESD is calculated as follows:

Values for ESD = values of PSD * Time Factor

were the Time Factor = (Block size)/Δf and Δf is the sampling rate / block size.

Notice that in **EU^2/Hz, or EU^{2}S/Hz**, EU really means the RMS unit of the EU, i.e., EU$_{rms}$.



Random signal with 1 volt RMS amplitude and Energy Spectrum Density format.

Cross Power Spectrum (CPS)

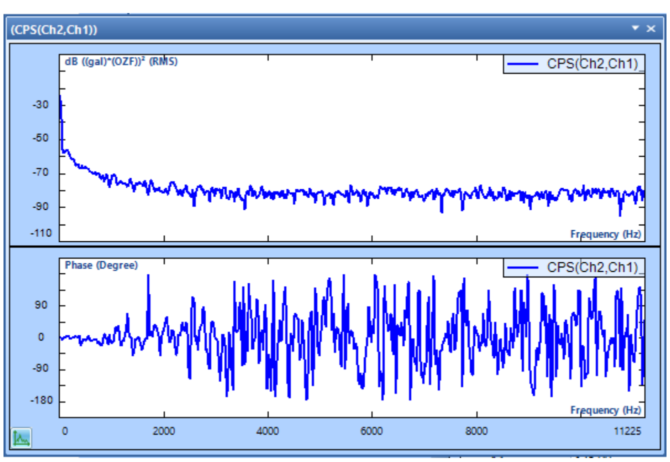
The Cross Spectrum characterizes the relationship between two spectra. For two signals x and y, with frequency components X(f)and Y(f), it is defined as:

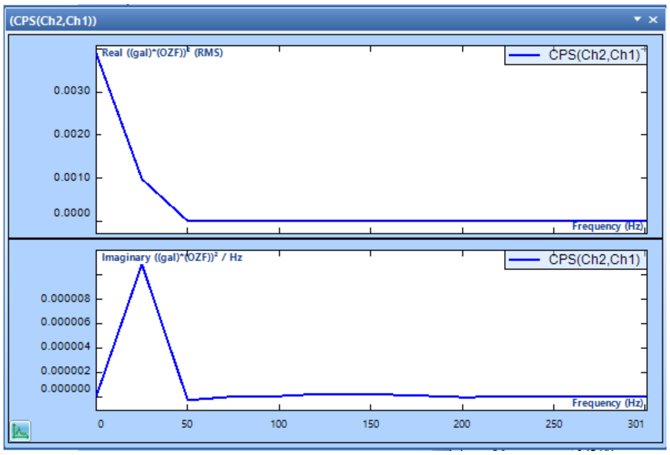
$$\boldsymbol{G_{xy}(f) = \frac{1}{N}\sum_{k=1}^{N} Y_k(f) X_k^*(f)}$$

The Cross Spectrum reflects the correlation between the two signals. While the Power Spectrum is real-valued, the Cross Spectrum is complex. This means that it also describes the phase relationship between the two signals.

Selecting different spectrum types will not affect the CPS spectrum in Real + Imaginary values.

A CPS signal from an EDM DSA FFT Analysis test:





(CPS(Ch2,Ch1))

	X Frequency (Hz)	CPS(Ch2,Ch1) Y-Re Real ((gal)*(OZF))^2 (RMS)	CPS(Ch2,Ch1) Y-Im Imaginary ((gal)*(OZF))^2 (RMS)
1	0.00	0.0039	0.0000
2	25.00	0.0010	1.0867E-005
3	50.00	2.8946E-006	-2.8693E-007
4	75.00	1.5555E-006	1.6748E-008
5	100.00	1.8282E-006	5.0550E-008
6	125.00	2.2522E-006	2.4669E-007
7	150.00	2.5861E-006	2.0777E-007
8	175.00	1.5724E-006	1.0243E-007
9	200.00	9.9860E-007	-3.7346E-008
10	225.00	7.1721E-007	-8.1721E-010
11	250.00	7.3486E-007	1.7514E-008
12	275.00	6.4370E-007	8.7293E-009
13	300.00	2.9944E-007	2.9369E-008
14	325.00	3.2161E-007	1.3775E-008
15	350.00	3.8622E-007	1.7962E-009

ATFX API C# Demo display

The ATFX API will read the CPS in Real & Imaginary values.

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)
CPS(Ch2,Ch1)
CPS(Ch3,Ch1)
CPS(Ch4,Ch1)
CPS(Ch5,Ch1)
H(Ch2,Ch1)
COH(Ch2,Ch1)
H(Ch3,Ch1)
COH(Ch3,Ch1)
H(Ch4,Ch1)
COH(Ch4,Ch1)
H(Ch5,Ch1)
COH(Ch5,Ch1)
FFT(Ch1)
FFT(Ch2)
FFT(Ch3)
FFT(Ch4)
FFT(Ch5)

X Data-Frequency (Hz)	Y Data-Real (gal)*(OZF)	Y data-Imaginary (gal)*(OZF)
0	0.00389975868165493	0
25	0.000978268450126052	1.086672909877...
50	2.89462786895456E-06	-2.86934437099...
75	1.55553834702005E-06	1.674782978966...
100	1.82822236638458E-06	5.055041896184...
125	2.25224448513472E-06	2.466854027716...
150	2.58609225056716E-06	2.077682097478...
175	1.57242186560325E-06	1.024331766075...
200	9.98601649371267E-07	-3.73458526325...
225	7.17210582479311E-07	-8.17212963966...
250	7.34857167117298E-07	1.751381262238...
275	6.43697944724408E-07	8.729297285015...
300	2.99443826179413E-07	2.936857512736...
325	3.21606762554438E-07	1.377534886159...

Frequency Response Function (FRF)

The cross-power spectrum method is used for estimating the frequency response function between channel x and channel y. The equation is:

$$Hyx = \frac{Gyx}{Gxx}$$

where *Gyx* is the averaged cross-spectrum between the input channel x and output channel y. *Gxx* is the averaged auto-spectrum of the input. Either power spectrum, power spectral density, or energy spectral density can be used here because of the linear relationship between input and output.

This approach will reduce the effect of the noise at the output measurement end, as shown below.

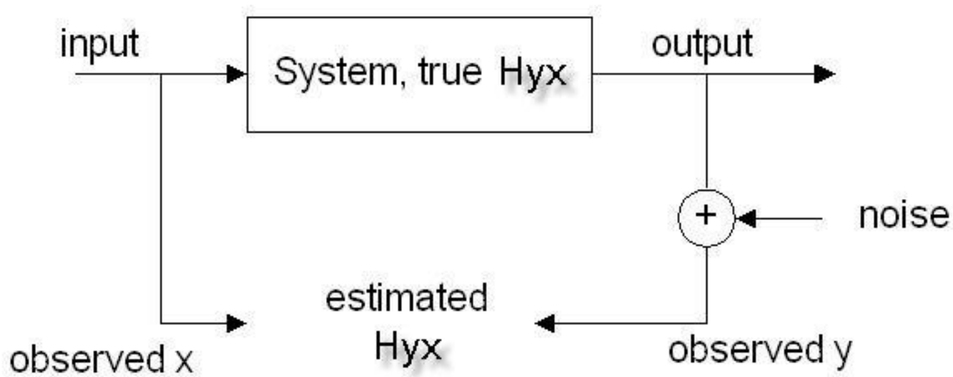


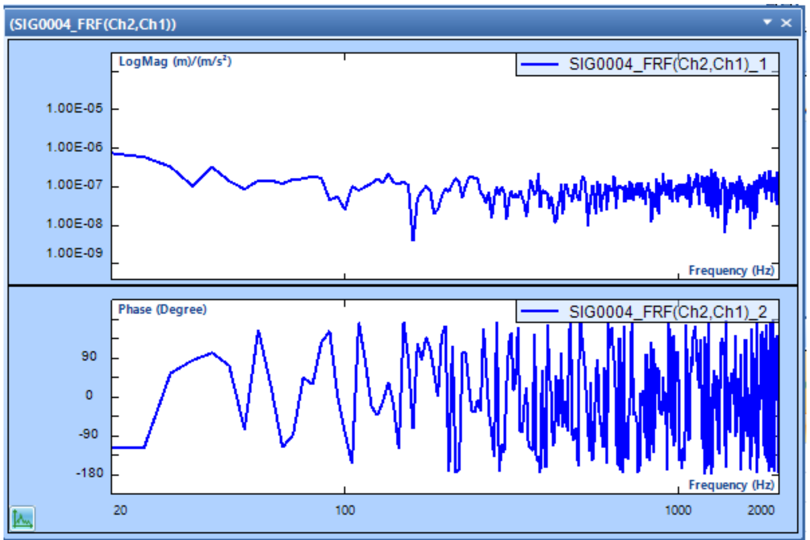
Figure 1. Frequency Response Function Computation

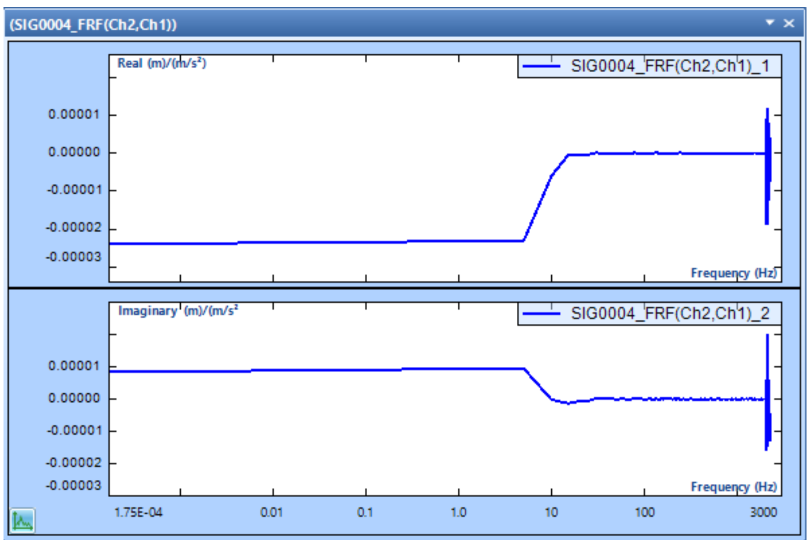
The frequency response function has a complex data format. You can view it in real, imaginary, magnitude, or phase display format.

Please note when describing a system with input x and output y as shown above, some people are used to a notation Hyx instead of Hxy. Most DSA products follow the convention used in the reference books listed before. Hxy stands for a frequency response function with input x and output y.

Selecting different spectrum types will not affect the FRF spectrum in Real + Imaginary values.

An FRF signal from an EDM VCS Random test:



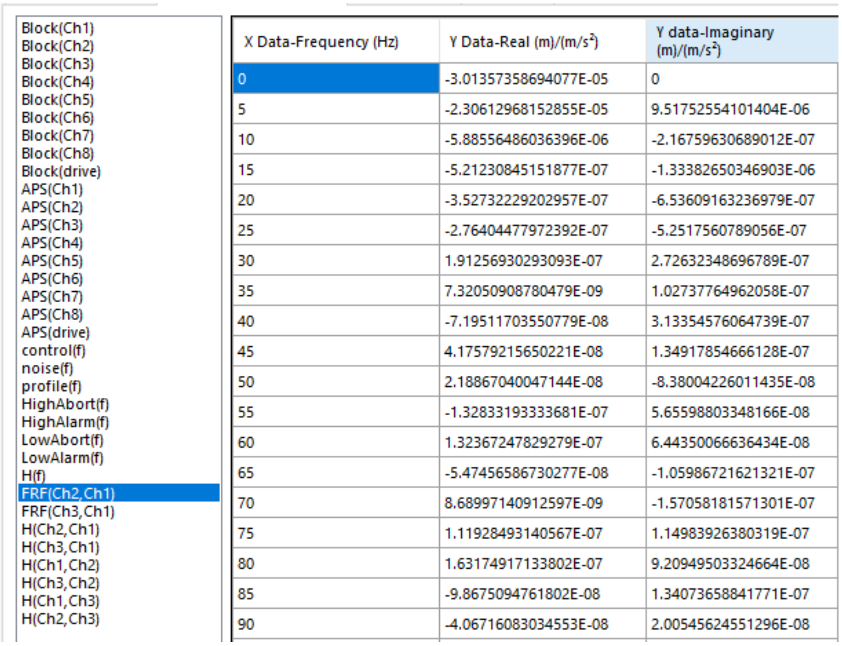


(SIG0004_FRF(Ch2,Ch1))

	X Frequency (Hz)	SIG0004_FRF(Ch2,Ch1 Y-Re Real (m)/(m/s²)	SIG0004_FRF(Ch2,Ch1 Y-Im Imaginary (m)/(m/s²)
1	0.00	-3.0136E-005	0.0000
2	5.00	-2.3061E-005	9.5175E-006
3	10.00	-5.8856E-006	-2.1676E-007
4	15.00	-5.2123E-007	-1.3338E-006
5	20.00	-3.5273E-007	-6.5361E-007
6	25.00	-2.7640E-007	-5.2518E-007
7	30.00	1.9126E-007	2.7263E-007
8	35.00	7.3205E-009	1.0274E-007
9	40.00	-7.1951E-008	3.1335E-007
10	45.00	4.1758E-008	1.3492E-007
11	50.00	2.1887E-008	-8.3800E-008
12	55.00	-1.3283E-007	5.6560E-008
13	60.00	1.3237E-007	6.4435E-008
14	65.00	-5.4746E-008	-1.0599E-007
15	70.00	8.6900E-009	[illegible]1.5706E-007

ATFX API C# Demo display

The ATFX API will read the FRF in Real & Imaginary values.



X Data-Frequency (Hz)	Y Data-Real (m)/(m/s²)	Y data-Imaginary (m)/(m/s²)
0	-3.01357358694077E-05	0
5	-2.30612968152855E-05	9.51752554101404E-06
10	-5.88556486036396E-06	-2.16759630689012E-07
15	-5.21230845151877E-07	-1.33382650346903E-06
20	-3.52732229202957E-07	-6.53609163236979E-07
25	-2.76404477972392E-07	-5.2517560789056E-07
30	1.91256930293093E-07	2.72632348696789E-07
35	7.32050908780479E-09	1.02737764962058E-07
40	-7.19511703550779E-08	3.13354576064739E-07
45	4.17579215650221E-08	1.34917854666128E-07
50	2.18867040047144E-08	-8.38004226011435E-08
55	-1.32833193333681E-07	5.65598803348166E-08
60	1.32367247829279E-07	6.44350066636434E-08
65	-5.47456586730277E-08	-1.05986721621321E-07
70	8.68997140912597E-09	-1.57058181571301E-07
75	1.11928493140567E-07	1.14983926380319E-07
80	1.63174917133802E-07	9.20949503324664E-08
85	-9.8675094761802E-08	1.34073658841771E-07
90	-4.06716083034553E-08	2.00545624551296E-08

Coherence Function (COH)

The coherence function is defined as:

$$C_{yx}{}^{2} = \frac{|G_{yx}|^{2}}{G_{xx}G_{yy}}$$

where *Gyx* is the averaged cross-spectrum between the input channel x and output channel y. *Gxx* and *Gyy* are the averaged auto-spectrum of the input and output. Either power spectrum, power spectral density, or energy spectral density can be used here because of the linear relationship between input and output.

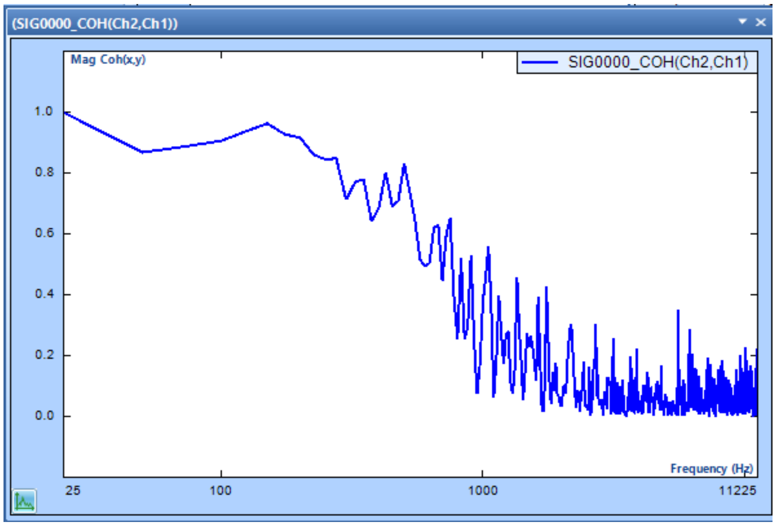
When the averaging number is 1, coherence function has a meaningless result of 1.0 due to the estimation error of the coherence function.

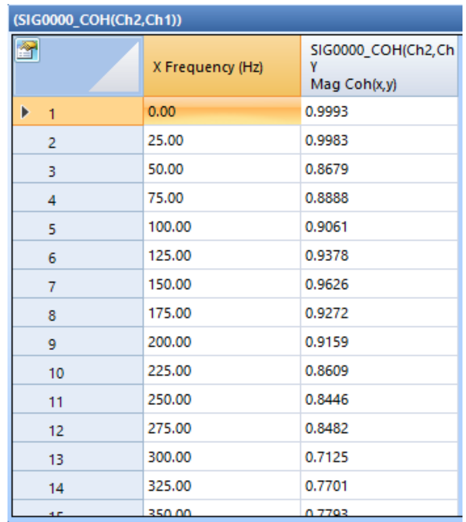
The coherence function is a non-dimensional real function in the frequency domain. It can only be viewed in the real format.

Please note when describing a system with input x and output y as shown above, some people are used to a notation Hyx instead of Hxy. Most DSA products follow the convention used in the reference books listed before. Hxy stands for a frequency response function with input x and output y.

Selecting different spectrum types will not affect the COH spectrum.

An COH signal from an EDM DSA FFT Analysis test:

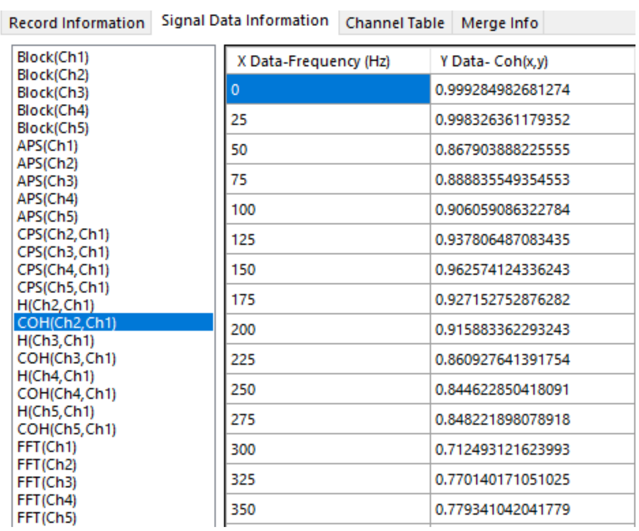




(SIG0000_COH(Ch2,Ch1))

	X Frequency (Hz)	SIG0000_COH(Ch2,Ch Y Mag Coh(x,y)
1	0.00	0.9993
2	25.00	0.9983
3	50.00	0.8679
4	75.00	0.8888
5	100.00	0.9061
6	125.00	0.9378
7	150.00	0.9626
8	175.00	0.9272
9	200.00	0.9159
10	225.00	0.8609
11	250.00	0.8446
12	275.00	0.8482
13	300.00	0.7125
14	325.00	0.7701
15	350.00	0.7793

ATFX API C# Demo display



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

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)
CPS(Ch2,Ch1)
CPS(Ch3,Ch1)
CPS(Ch4,Ch1)
CPS(Ch5,Ch1)
H(Ch2,Ch1)
COH(Ch2,Ch1)
H(Ch3,Ch1)
COH(Ch3,Ch1)
H(Ch4,Ch1)
COH(Ch4,Ch1)
H(Ch5,Ch1)
COH(Ch5,Ch1)
FFT(Ch1)
FFT(Ch2)
FFT(Ch3)
FFT(Ch4)
FFT(Ch5)

X Data-Frequency (Hz)	Y Data- Coh(x,y)
0	0.999284982681274
25	0.998326361179352
50	0.867903888225555
75	0.888835549354553
100	0.906059086322784
125	0.937806487083435
150	0.962574124336243
175	0.927152752876282
200	0.915883362293243
225	0.860927641391754
250	0.844622850418091
275	0.848221898078918
300	0.712493121623993
325	0.770140171051025
350	0.779341042041779

Sine Spectrum

Spectrum is the sine measurement value plotted across the frequency. Usually it is represented in acceleration peak value. The sine measurement is taken at the output of tracking filter. The spectrum in sine is not the FFT transform of a time measurement. It is just the history trace of equivalent sine peak values drawn across the whole frequency. The resolution of spectrum signal has nothing to do with the resolution of frequency change in the control process.

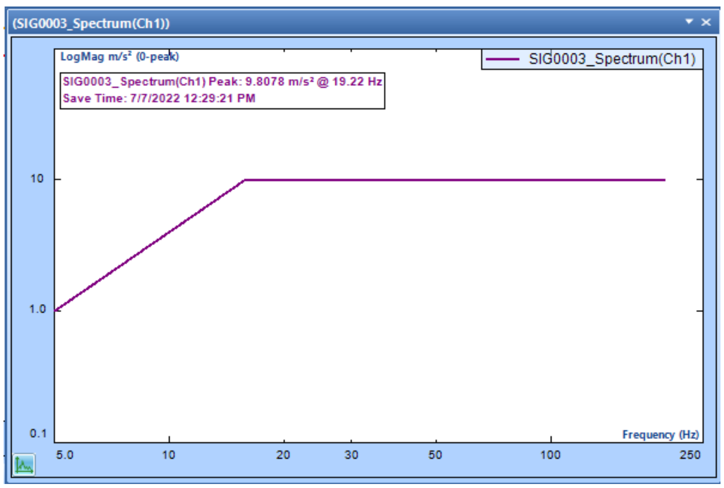
The magnitude of the frequency components of signals are collectively called the amplitude spectrum. In many applications, the quantity of interest is the power or the rate of energy transfer that is proportional to the squared magnitude of the frequency components. The average squared magnitudes of all the DFT frequency lines are collectively referred to as the Power Spectrum, G_{xx}.

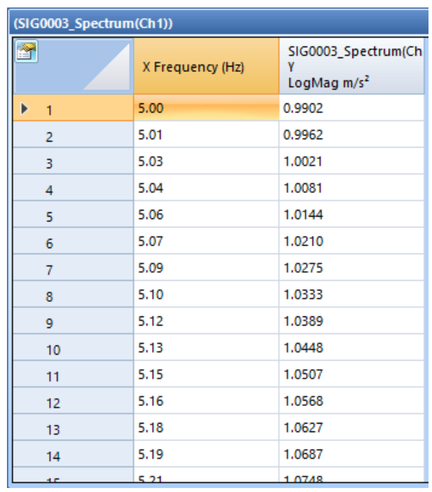
The averaging process is more properly termed an ensemble average, wherein the squared amplitude from N signal blocks at each measured frequency, f, are averaged together. Letting an asterisk (*) denote conjugation of a complex number, the "power" averaging process is defined by:

$$\boldsymbol{G_{xx}(f) = |X(f)|^2 = \frac{1}{N}\sum_{k=1}^{N} X_k(f)X_k^*(f)}$$

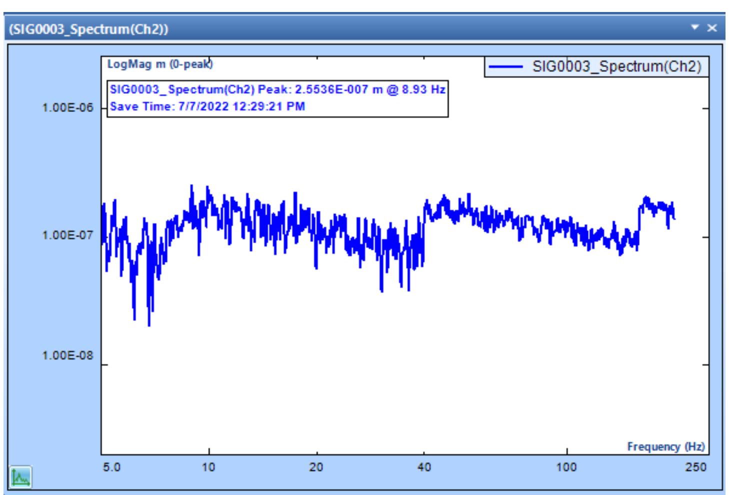
Selecting different spectrum types will affect the Sine spectrum.

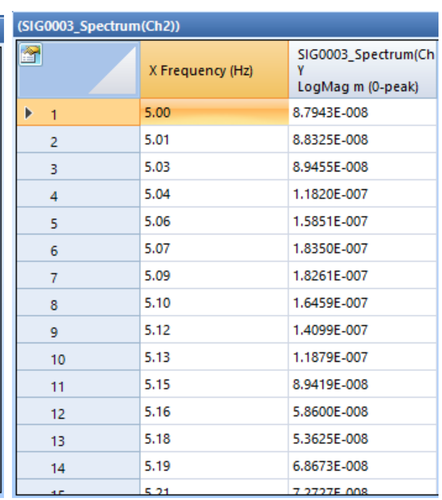
Two Sine spectrum signals from an EDM VCS Swept Sine test:




(SIG0003_Spectrum(Ch1))

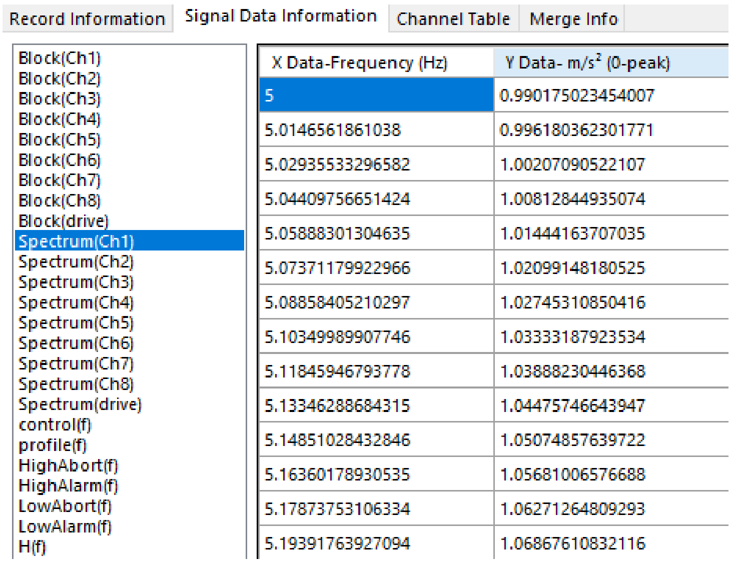
	X Frequency (Hz)	SIG0003_Spectrum(Ch Y LogMag m/s²
1	5.00	0.9902
2	5.01	0.9962
3	5.03	1.0021
4	5.04	1.0081
5	5.06	1.0144
6	5.07	1.0210
7	5.09	1.0275
8	5.10	1.0333
9	5.12	1.0389
10	5.13	1.0448
11	5.15	1.0507
12	5.16	1.0568
13	5.18	1.0627
14	5.19	1.0687



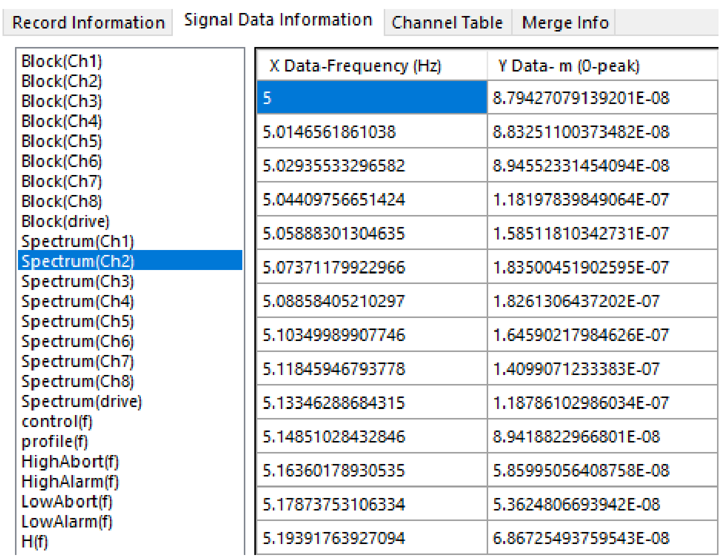

(SIG0003_Spectrum(Ch2))

	X Frequency (Hz)	SIG0003_Spectrum(Ch Y LogMag m (0-peak)
1	5.00	8.7943E-008
2	5.01	8.8325E-008
3	5.03	8.9455E-008
4	5.04	1.1820E-007
5	5.06	1.5851E-007
6	5.07	1.8350E-007
7	5.09	1.8261E-007
8	5.10	1.6459E-007
9	5.12	1.4099E-007
10	5.13	1.1879E-007
11	5.15	8.9419E-008
12	5.16	5.8600E-008
13	5.18	5.3625E-008
14	5.19	6.8673E-008

ATFX API C# Demo display



X Data-Frequency (Hz)	Y Data- m/s² (0-peak)
5	0.990175023454007
5.0146561861038	0.996180362301771
5.02935533296582	1.00207090522107
5.04409756651424	1.00812844935074
5.05888301304635	1.01444163707035
5.07371179922966	1.02099148180525
5.08858405210297	1.02745310850416
5.10349989907746	1.03333187923534
5.11845946793778	1.03888230446368
5.13346288684315	1.04475746643947
5.14851028432846	1.05074857639722
5.16360178930535	1.05681006576688
5.17873753106334	1.06271264809293
5.19391763927094	1.06867610832116



X Data-Frequency (Hz)	Y Data- m (0-peak)
5	8.79427079139201E-08
5.0146561861038	8.83251100373482E-08
5.02935533296582	8.94552331454094E-08
5.04409756651424	1.18197839849064E-07
5.05888301304635	1.58511810342731E-07
5.07371179922966	1.83500451902595E-07
5.08858405210297	1.8261306437202E-07
5.10349989907746	1.64590217984626E-07
5.11845946793778	1.4099071233383E-07
5.13346288684315	1.18786102986034E-07
5.14851028432846	8.9418822966801E-08
5.16360178930535	5.85995056408758E-08
5.17873753106334	5.3624806693942E-08
5.19391763927094	6.86725493759543E-08

Shock Response Spectrum (SRS)

The Shock Response Spectrum (SRS) is an entirely different type of spectral measurement. It is used to access the damage potential of a transient event such as a package drop or an earthquake. The SRS was first proposed by Dr. Maurice Biot in 1932.

The SRS is not the spectrum of the pulse. (The FFT provides this.) The SRS is not a linear operator as the FFT is. That is, an SRS does not uniquely define a single waveform. Many very different transient time-histories can produce the same SRS.

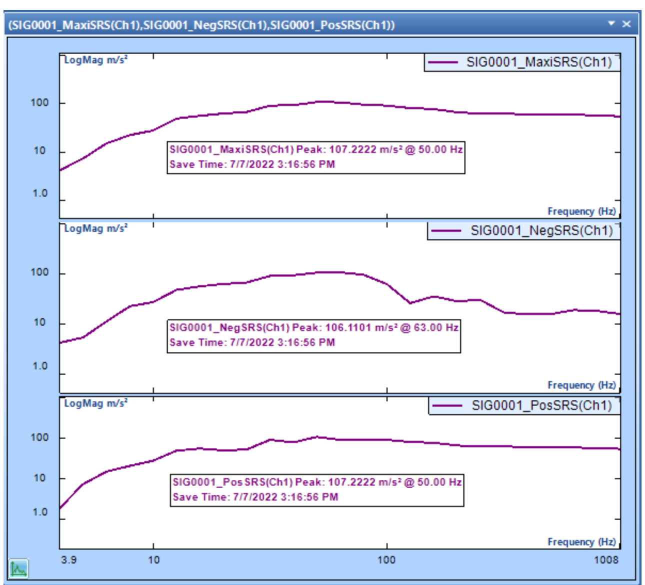
What the Shock Response Spectrum is, is the representative response of a class of simple structures to the given transient acceleration time-history. This response is provided by simulating a group of spring-mass-damper systems sitting on a common rigid base that is forced to move with the measured acceleration of the subject shock pulse. Each single degree-of-freedom (SDOF) spring-mass-damper has a different natural frequency; they all have the same damping factor. The spectrum is formed by plotting the extreme motion (acceleration) experienced by each mass against its resonance frequency.

The frequency spacing of the resonance frequencies is logarithmic, much like the 1/3 octave filters used in acoustical analysis. That is, it is a type of proportional bandwidth analysis where the half-power bandwidth of each SDOF system increases in proportion to its resonance frequency. The resolution of an SRS is defined by the number of simulated SDOFs included in the desired analysis span. The percent damping of all the SDOFs is selectable (although most tests specify 5% damping).

The extreme motion of each mathematically simulated SDOF mass is monitored by several peak detectors. The extreme positive and negative accelerations are retained *during the duration of the input pulse* and *after it*. Maximum and minimum values captured during the pulse's duration are termed *Primary* extremes. Those found after the pulse has returned to zero are termed *Residual* extremes. Specific tests will prescribe whether positive, negative, or extreme absolute values captured should be displayed. They will further specify Primary, Residual, or combined (maxi-max) data be plotted.

Selecting different spectrum types will not affect the SRS spectrum.

The Maxi, Pos, and Neg SRS signals from an EDM VCS Shock test:



(SIG0001_MaxiSRS(Ch1))

	X Frequency (Hz)	SIG0001_MaxiSRS(Ch1 Y LogMag m/s²
1	3.94	4.2063
2	4.96	7.3834
3	6.25	15.0472
4	7.87	22.3844
5	9.92	27.8444
6	12.50	49.0164
7	15.75	55.8369
8	19.84	62.2705
9	25.00	66.6374
10	31.50	90.7439
11	39.69	92.5792
12	50.00	107.2222
13	63.00	106.1101
14	79.37	95.0206
15	100.00	90.7022

(SIG0001_PosSRS(Ch1))

	X Frequency (Hz)	SIG0001_PosSRS(Ch1) Y LogMag m/s²
1	3.94	1.9085
2	4.96	7.3834
3	6.25	15.0472
4	7.87	20.7145
5	9.92	27.8444
6	12.50	49.0164
7	15.75	55.3370
8	19.84	49.8204
9	25.00	52.3731
10	31.50	90.7439
11	39.69	78.2693
12	50.00	107.2222
13	63.00	90.6851
14	79.37	93.4566
15	100.00	90.7022

(SIG0001_NegSRS(Ch1))

	X Frequency (Hz)	SIG0001_NegSRS(Ch1) Y LogMag m/s²
1	3.94	4.2063
2	4.96	5.3528
3	6.25	11.1116
4	7.87	22.3844
5	9.92	27.1121
6	12.50	47.4811
7	15.75	55.8369
8	19.84	62.2705
9	25.00	66.6374
10	31.50	89.8160
11	39.69	92.5792
12	50.00	104.7085
13	63.00	106.1101
14	79.37	95.0206
15	100.00	62.2454

ATFX API C# Demo display

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

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
Block(Ch6)
Block(Ch7)
Block(Ch8)
profile(t)
profile(f)
HighAbort(t)
LowAbort(t)
Block(drive)
drive(f)
control(t)
control(f)
noise(t)
hinv(f)
error_t
APS(Ch1)
APS(Ch2)
MaxiSRS(Ch1)
PosSRS(Ch1)
NegSRS(Ch1)

X Data-Frequency (Hz)	Y Data-m/s²
3.93725380633059	4.20627546310425
4.96062894937461	7.38337087631226
6.25000083403495	15.0472183227539
7.87450761266112	22.3844356536865
9.92125789874915	27.8443756103516
12.5000016680698	49.0164337158203
15.7490152253221	55.8368759155273
19.8425157974982	62.2705116271973
25.0000033361394	66.6374053955078
31.498030450644	90.7438888549805
39.685031594996	92.5791854858398
50.0000066722785	107.222160339355
62.9960609012876	106.110076904297

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

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
Block(Ch6)
Block(Ch7)
Block(Ch8)
profile(t)
profile(f)
HighAbort(t)
LowAbort(t)
Block(drive)
drive(f)
control(t)
control(f)
noise(t)
hinv(f)
error_t
APS(Ch1)
APS(Ch2)
MaxiSRS(Ch1)
PosSRS(Ch1)
NegSRS(Ch1)

X Data-Frequency (Hz)	Y Data-m/s²
3.93725380633059	1.90849030017853
4.96062894937461	7.38337087631226
6.25000083403495	15.0472183227539
7.87450761266112	20.714506149292
9.92125789874915	27.8443756103516
12.5000016680698	49.0164337158203
15.7490152253221	55.3370170593262
19.8425157974982	49.8204383850098
25.0000033361394	52.3731155395508
31.498030450644	90.7438888549805
39.685031594996	78.2693405151367
50.0000066722785	107.222160339355
62.9960609012876	90.6851196289063

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

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
Block(Ch6)
Block(Ch7)
Block(Ch8)
profile(t)
profile(f)
HighAbort(t)
LowAbort(t)
Block(drive)
drive(f)
control(t)
control(f)
noise(t)
hinv(f)
error_t
APS(Ch1)
APS(Ch2)
MaxiSRS(Ch1)
PosSRS(Ch1)
NegSRS(Ch1)

X Data-Frequency (Hz)	Y Data-m/s²
3.93725380633059	4.20627546310425
4.96062894937461	5.35278511047363
6.25000083403495	11.1116371154785
7.87450761266112	22.3844356536865
9.92125789874915	27.1121196746826
12.5000016680698	47.4811096191406
15.7490152253221	55.8368759155273
19.8425157974982	62.2705116271973
25.0000033361394	66.6374053955078
31.498030450644	89.8159942626953
39.685031594996	92.5791854858398
50.0000066722785	104.708518981934
62.9960609012876	106.110076904297

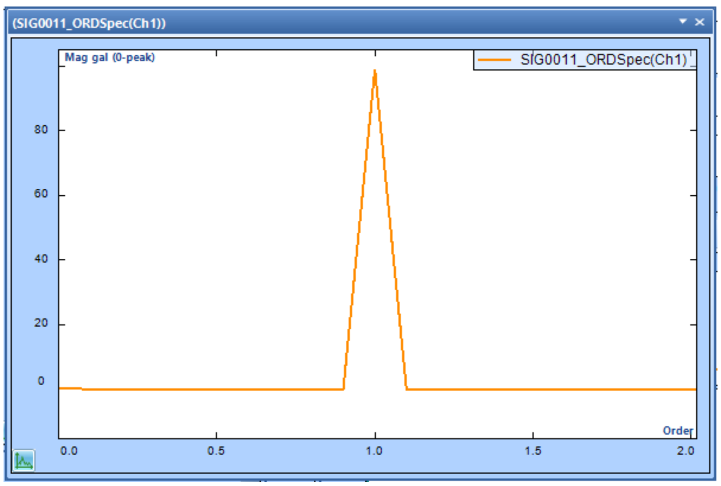
Order Spectrum

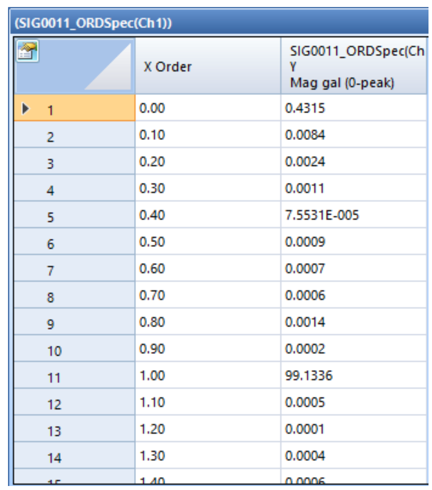
Synchronizing the sampling to the rotating speed allows presentation of measurement results in the angle and order domains in lieu of the time and frequency domains. An order is simply a frequency divided by a reference frequency, normally a machine's shaft-turning frequency. This means that the order location in an order-normalized spectrum indicates the number of vibration cycles per shaft revolution. The tracked magnitude (which can be measured using EU_{pk}, EU_{rms}, or EU_{rms}^2) of an order is the measurement extracted through a tracking filter with its center frequency located at this order.

An Order Power Spectrum measurement gives a quantitative description of the amplitude, or power, of the orders in a signal. It provides a good view of all order components of a signal. This can help you rapidly identify significant forcing mechanisms.

Selecting different spectrum types will affect the Order spectrum.

An order spectrum signal from an EDM DSA Order Tracking test:

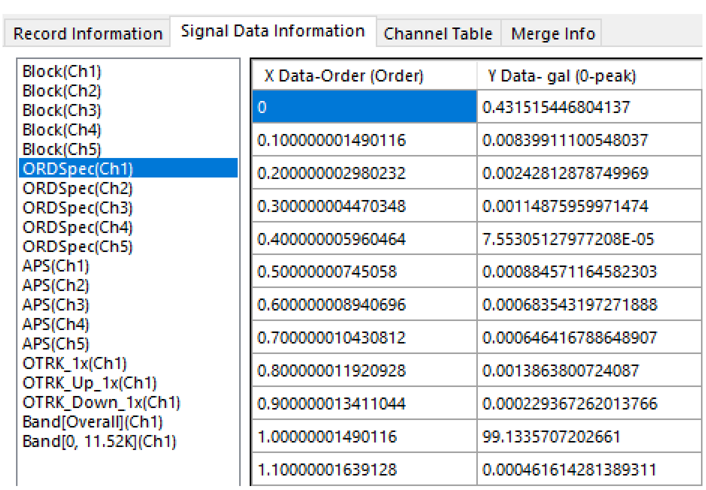




(SIG0011_ORDSpec(Ch1))

	X Order	SIG0011_ORDSpec(Ch Y Mag gal (0-peak)
1	0.00	0.4315
2	0.10	0.0084
3	0.20	0.0024
4	0.30	0.0011
5	0.40	7.5531E-005
6	0.50	0.0009
7	0.60	0.0007
8	0.70	0.0006
9	0.80	0.0014
10	0.90	0.0002
11	1.00	99.1336
12	1.10	0.0005
13	1.20	0.0001
14	1.30	0.0004
15	1.40	0.0006

ATFX API C# Demo display



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

Block(Ch1)
Block(Ch2)
Block(Ch3)
Block(Ch4)
Block(Ch5)
ORDSpec(Ch1)
ORDSpec(Ch2)
ORDSpec(Ch3)
ORDSpec(Ch4)
ORDSpec(Ch5)
APS(Ch1)
APS(Ch2)
APS(Ch3)
APS(Ch4)
APS(Ch5)
OTRK_1x(Ch1)
OTRK_Up_1x(Ch1)
OTRK_Down_1x(Ch1)
Band[Overall](Ch1)
Band[0, 11.52K](Ch1)

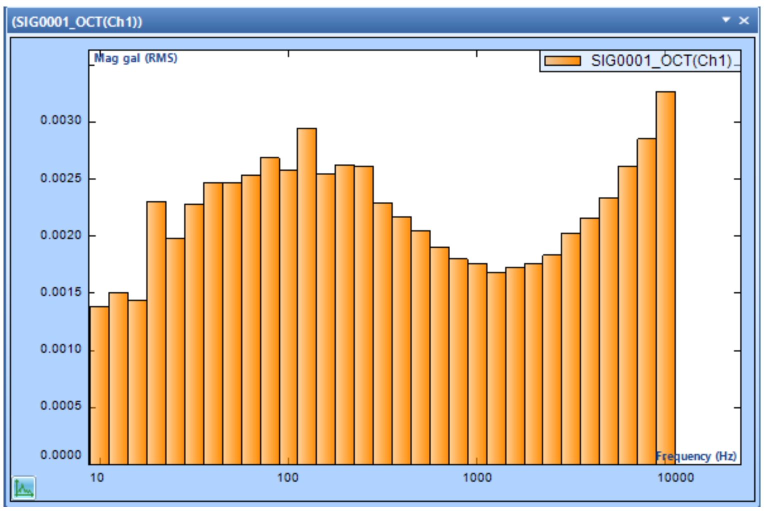
X Data-Order (Order)	Y Data- gal (0-peak)
0	0.431515446804137
0.100000001490116	0.00839911100548037
0.200000002980232	0.00242812878749969
0.300000004470348	0.00114875959971474
0.400000005960464	7.55305127977208E-05
0.50000000745058	0.000884571164582303
0.600000008940696	0.000683543197271888
0.700000010430812	0.000646416788648907
0.800000011920928	0.0013863800724087
0.900000013411044	0.000229367262013766
1.00000001490116	99.1335707202661
1.10000001639128	0.000461614281389311

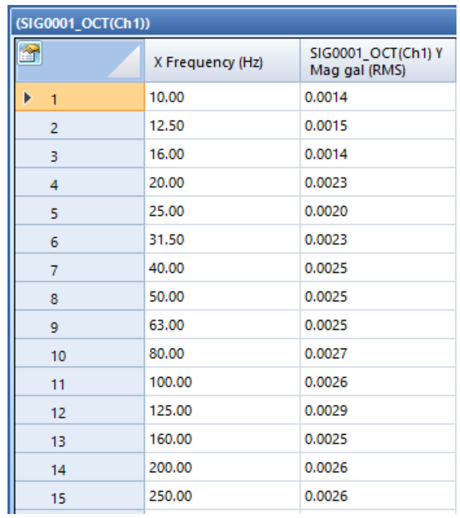
Octave Spectrum

The Fractional Octave Filter Analysis function applies a bank of real-time 1/nth octave filters to the input time streams and generates two types of responses at the same time: 1/Nth octave spectra, and the RMS time history of each 1/Nth octave filter band. The output of each real-time filter bank is in fact a 3D waterfall signal that is arranged with the x-axis as logarithmic frequency and the z-axis as time. Frequency weighting is applied in the frequency axis and time-weighting is applied in the time axis.

Selecting different spectrum types will affect the Octave spectrum.

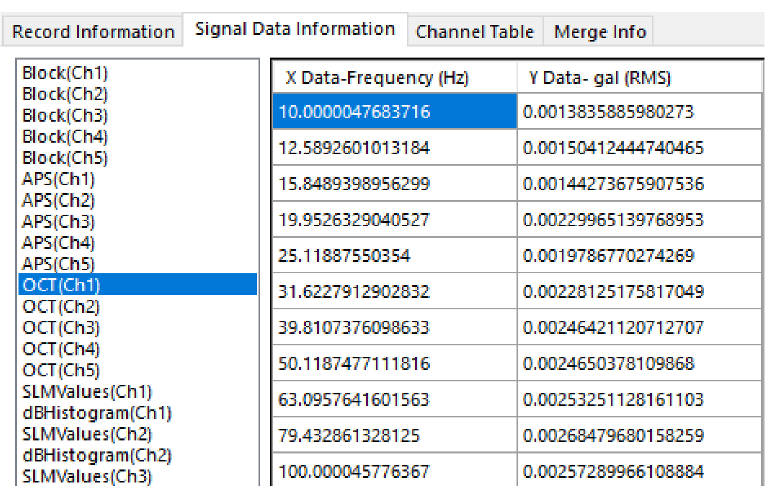
An octave signal from an EDM DSA Acoustic Analysis test:




(SIG0001_OCT(Ch1))

	X Frequency (Hz)	SIG0001_OCT(Ch1) Y Mag gal (RMS)
1	10.00	0.0014
2	12.50	0.0015
3	16.00	0.0014
4	20.00	0.0023
5	25.00	0.0020
6	31.50	0.0023
7	40.00	0.0025
8	50.00	0.0025
9	63.00	0.0025
10	80.00	0.0027
11	100.00	0.0026
12	125.00	0.0029
13	160.00	0.0025
14	200.00	0.0026
15	250.00	0.0026

ATFX API C# Demo display



X Data-Frequency (Hz)	Y Data- gal (RMS)
10.0000047683716	0.0013835885980273
12.5892601013184	0.00150412444740465
15.8489398956299	0.00144273675907536
19.9526329040527	0.00229965139768953
25.11887550354	0.0019786770274269
31.6227912902832	0.00228125175817049
39.8107376098633	0.00246421120712707
50.1187477111816	0.0024650378109868
63.0957641601563	0.00253251128161103
79.432861328125	0.00268479680158259
100.000045776367	0.00257289966108884

Compution of Frequency Spectrum Signals

Linear Spectrum

Most DSA products use the following steps to compute a linear spectrum:

Step 1

First a window is applied:

$$x(t) = w(t)\, x(t)'$$

where *x(t)'* is the original data and *x(t)* is the data used for the Fourier transform.

Step 2

The FFT is applied to *x(t)* to compute *X(k)*, as described above.

Step 3

Averaging is applied to *X(k)*. Here Averaging can be either an Exponential Average or Stable Average. Result is *Sx'*.

$$Sx' = Average\ (\ X(k)\ )$$

Step 4

To get a single-sided spectrum, double the value for symmetry about DC.

Amplitude Correction factor is applied to *Sx'* so that the result has an un-biased reading at the harmonic frequencies.

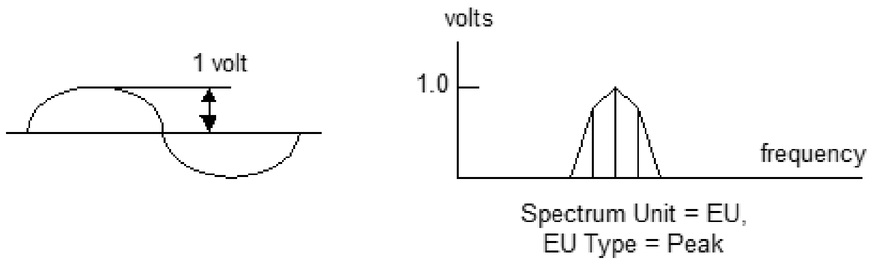
$$Sx = 2 \cdot Sx' / AmpCorr$$

where AmpCorr is the amplitude correction factor, defined as:

$$AmpCorr = \sum_{k=0}^{N-1} w(k)$$

where *w(k)* is the window weighting function.

This correction will make the reading at specific frequency correct even when a window is applied. For example, if a 1-volt amplitude sine wave is analyzed by Linear Spectrum with Hann window, you will get the following spectral shape:



Linear Spectrum of 1-Volt Sine Wave

Auto Power Spectrum

To compute the auto power spectra, the instrument will follow these steps:

Step 1

A window is applied:

$$\mathbf{x(k) = w(k)\ x(k)'}$$

where **x(k)’** is the original data and **x(k)** is the data used for a Fourier transform.

Step 2

The FFT is applied to **x(t)** to compute **Sx**

$$Sx = \sum_{n=0}^{N-1} x(k)\, e^{-j2\pi kn/N}$$

Next the so called periodogram method is used to compute the spectra with area correction. Using Sx.

Step 3

Calculate the Power Spectrum Sxx = Sx Sx* / $(\text{AmpCorr})^2$

Or calculate the Power Spectral Density = Sx Sx* T / EnergyCorr

Or calculate the Energy Spectral Density = Sx Sx* T2 / EnergyCorr

where T is the time duration of the capture. The symbol * is for complex conjugation. EnergyCorr is a factor for energy correction, which is defined as:

$$EnergyCorr = \frac{1}{N}\sum_{k=0}^{N-1} w(k)^2$$

N is the total number of the samples and w(k) is window function.

For any power spectral measurement of the three types listed above, the EU is automatically chosen as EU_{rms} because only EU_{rms} has a physical meaning related to signal power.

After the power spectra are calculated, the averaging operation will be applied.

Cross Power Spectrum

To compute the cross-power spectral density Gyx between channel x and channel y:

Step 1

Compute the Fourier transform of input signal x(k) and response signal y(k):

$$Sx = \sum_{n=0}^{N-1} x(k)\ w(k)\, e^{-j2\pi kn/N}$$

$$Sy = \sum_{n=0}^{N-1} y(k)\ w(k)\, e^{-j2\pi kn/N}$$

Step 2

Compute the instantaneous cross power spectral density:

$$Syx = Sx^{*}\ Sy\ T$$

Step 3

Average the M frames of Sxx to get averaged PSD Gxx

$$Gyx' = \text{Average}\ (Syx)$$

Step 4

Compute the energy correction and double the value for the single-sided spectra

$$Gyx = 2\ Gyx' / \text{EnergyCorr}$$

Frequency Response Function

An important application of Dynamic Signal Analysis is characterizing the input-output behavior of physical systems. In linear systems, the output can be predicted from a known input if the Frequency Response Function (FRF) of the system is known. The Frequency Response Function, H(f), relates the Fourier Transform of the input X(f) to the Fourier Transform of the output Y(f) by the simple equation:

$$\boldsymbol{Y(f) = H_{xy}(f)X(f)}$$

Multiplying both sides of this equation by the conjugate of the input spectrum and ensemble averaging explains the importance of the power and cross power spectra as they allow H(f) to be measured and calculated.

$$\boldsymbol{\frac{1}{N}\sum_{k=1}^{N} Y_k(f)X_k^*(f) = G_{xy}(f) = H_{xy}(f)\frac{1}{N}\sum_{k=1}^{N} X_k(f)X_k^*(f) = H_{xy}(f)G_{xx}(f)}$$

That is:

$$\boldsymbol{H_{xy}(f) = \frac{G_{xy}(f)}{G_{xx}(f)}}$$

The fact that Y(f) is dependent on the input X(f) is what makes the system linear. When measuring the input-output behavior of a system, there is always noise present that obscures the output. An important measure is how much of the output is actually caused by the input *and a linear process*. This is indicated by another important real-valued spectrum called the (ordinary) Coherence Function. This coherence function is also defined in terms of the cross spectrum and the power spectra. Specifically:

$$\boldsymbol{\gamma_{xy}^2(f) = \frac{G_{xy}(f)G_{xy}^*(f)}{G_{xx}(f)G_{yy}(f)}}$$

Note that the coherence can also be stated as the product of an FRF with its inverse function. That is, if H_{xy} measures a process going from input, x, to output, y, H_{yx} characterizes the same process, but treats y as the input and x as the output.

$$\boldsymbol{\gamma_{xy}^2(f) = H_{xy}(f)\frac{G_{xy}^*}{G_{yy}} = H_{xy}(f)H_{yx}(f)}$$

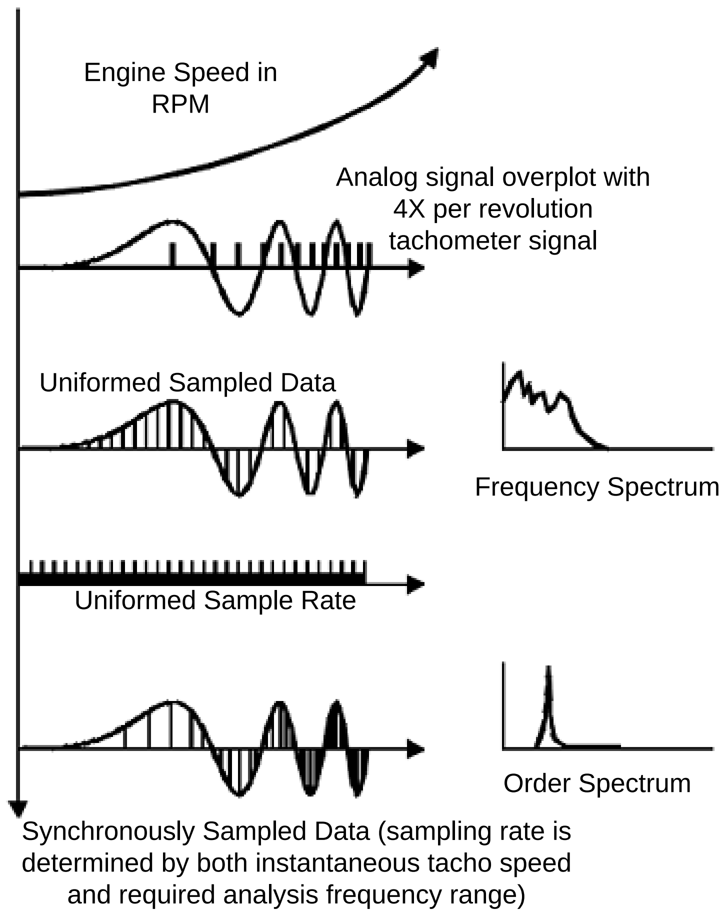
This product definition indicates the coherence represents an "energy round trip" or a reflection through the process. We apply G_{xx} to H_{xy} and get G_{xy} at the output. Then we conjugate G_{xy} (to flip it or reflect x(t) in time) and pass it through H_{yx}. In a perfect world, this would result in exactly G_{xx} as the output of H_{yx}.

If the system is linear and none of our measurements are contaminated by noise, the trip is perfect, and we get back everything we put in. That is, the coherence will be exactly 1.0. If the system is non-linear or if extraneous noise has been interjected, the round-trip will be less efficient, and the coherence will be less than one (but never more).

Thus, the coherence is always between 0 and 1. A coherence of 1.0 means the output is perfectly explained by the input (i.e., the system is linear). A coherence of 0 means the output and input are unrelated. Values in-between state the fraction of measured output power explained by the measured input power and a linear process. Experienced analysts always use the coherence measurement to quantify the quality of an FRF measurement at every frequency.

Order Spectrum

The following figure shows conceptually how angle re-sampling can be used to analyze vibrations from an engine during start up. Once the signal has been transformed into its angle domain, the FFT can be applied to analyze the order spectrum of the vibrations.

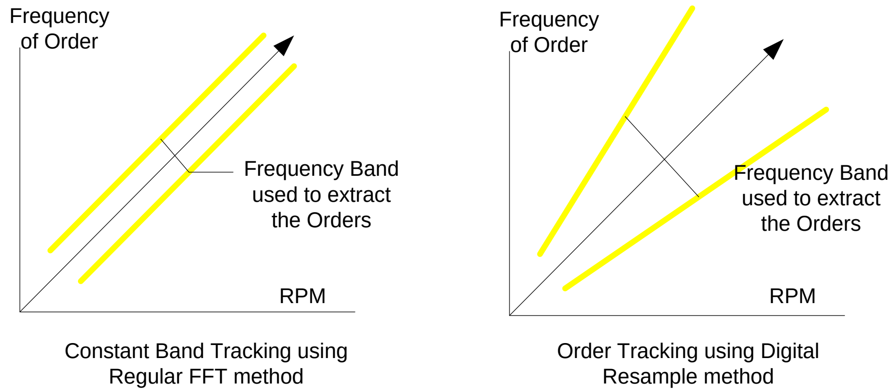


Angular data resampling of a chirp signal

An important concept that must be introduced now is called ΔOrder (delta order). In the FFT based frequency spectrum analysis, the frequency span and frequency resolution are fixed. The capability of discriminating frequency components is equal in both low and high frequency. In rotating machine analysis, we need to have better analysis resolution in the low frequency than that in high frequency.

For example, if the rotating speed is at 60 RPM, we care if the instrument can tell the difference between 1Hz (order 1) and 2Hz (order 2); in contrast, if the rotating speed is at 6000 RPM, the user probably will not care if the instrument can discriminate the measurement between 100Hz (order 1) and 101Hz.

With the digital resampling technique, the order tracks and order spectrum are extracted based on a filter with equal ΔOrder instead of equal ΔFrequency. The concept is illustrated in the following figure:



Comparison of constant band tracking and digital re-sampling method

The left figure shows when the order tracks are extracted using conventional FFT method with fixed resolution, the ΔFrequency of the tracking filter will be fixed; the right figure illustrates that if the order tracks are extracted using digital resampling, the ΔFrequency tracking filter will be increased proportionally with the RPM. Obviously, the method of digital resampling is more desirable in extracting the measurement of orders.

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 receives 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 REMEDIED 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