

# Keysight Technologies

## Easily Create Custom Waveform Plug-Ins With Waveform Creator Application Software



### Application Note

Enable higher productivity through a simple, open and expandable waveform creation environment. Create and deploy complex waveforms or develop emerging communications standards based waveforms.



## Introduction

The Keysight Technologies, Inc. M9099 Waveform Creator application software provides a simple, open and expandable waveform creation environment for use with Keysight's M9381A PXIe Vector Signal Generator and SystemVue system level design software. Designing devices for higher data rates and wider bandwidths requires more complex signals to verify how the device would perform in its real environment. Using Waveform Creator, individual waveform segments can be created using available waveform plug-ins or user created plug-ins. This application note describes how to create user plug-ins which can be fully integrated with Waveform Creator to deploy signal generation capabilities that meet your requirements.

The Keysight Waveform Creator provides a framework for creating custom baseband and modulated RF waveforms and enables the custom waveforms to be used in R&D, design validation and manufacturing test. This paper describes how waveform segments can be created, dragged and modified with user-definable parameters and then downloaded into the M9381A PXIe VSG or outputted as an unencrypted data file.

## Key issues faced in complex signal generation

Test engineers face the challenge of creating complex custom signals needed to test many of today's RF systems and emerging communications standards. Typically, assembly of such waveforms requires multiple tools and different waveform formats to be aggregated. This leaves the user with the time consuming task of modifying waveform timing alignment, resampling to different carrier frequencies and sample rates, as well as performing waveform validation.

This application note demonstrates how to use Waveform Creator, a simple, yet powerful tool to address these challenges. Waveform Creator provides:

- A simple, standard programming interface to easily create user definable and configurable waveform plug-ins.
- A platform for using customized plug-ins to create custom waveforms.

Waveform Creator ships with generic plug-ins, implemented with a selection of digital modulation formats that can also be manipulated as described above. Waveforms generated by Keysight's SystemVue can also be accessed and manipulated by Waveform Creator.

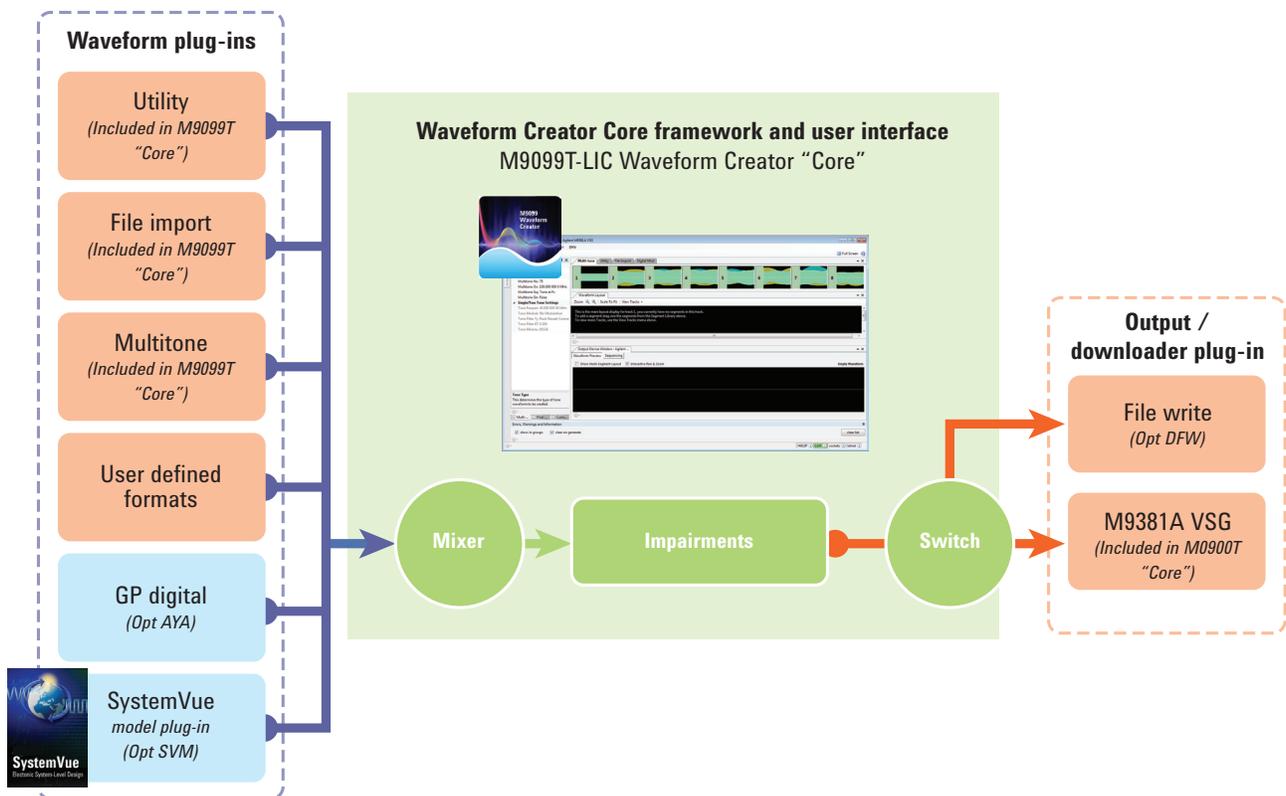


Figure 1. Plug-ins shipped with Waveform Creator indicated by red shaded boxes, including user defined format capability, the subject of this application note.

# Creating custom waveforms with Waveform Creator

The M9099 Waveform Creator application software enables custom functions, implemented as external DLLs, to be created and “plugged into” a general purpose core with very little system programming overhead. These custom plug-ins can be used to implement in-house proprietary technology, that can then be distributed as a reference implementation to other Waveform Creator installations within an organization. Once a waveform is generated by a plug-in, it can be mixed with other signals, deliberately distorted, post processed, have noise added, be  $\sin(x)/x$  pre-corrected, then saved to a file for use in simulation work, or downloaded to a Keysight instrument, such as a vector signal generator, for real-time play out.

## Building a custom waveform plug-in

This section demonstrates how easy it is to create a plug-in that generates a simple FM waveform, at IQ baseband, with control over the modulating frequency and the modulation index. The process illustrated below uses Microsoft Visual Studio 2010 (“VS2010”) as the development environment.

1. Run VS2010 and from the menu bar select File | New | Project...
2. Select Class Library from the list of templates and enter a project name. In the example, it is called “MyWaveform”. See Figure 2.

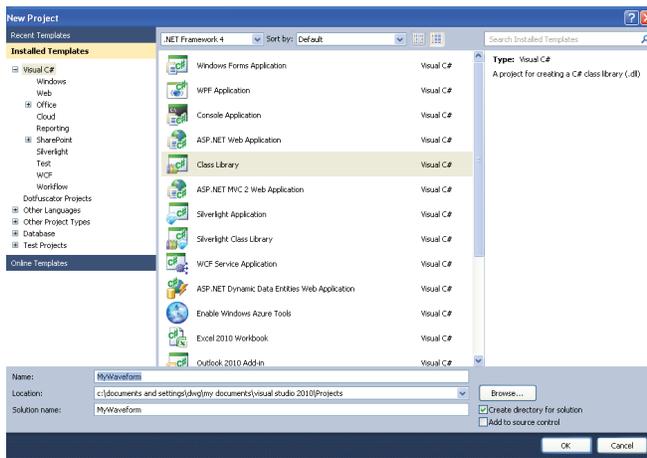


Figure 2. New project setup in Microsoft Visual Studio 2010.

3. VS2010 will construct the project and open the Class1.cs source file containing some basic template code.
4. In the Solution Explorer pane, right click on the default class name “Class1.cs” and rename. In the example, it is named “SimpleFM.cs”. VS2010 will ask to perform a rename of all references to the code element ‘Class 1’. Select “Yes”.
5. Again in the Solution Explorer pane, right click on “References” and select “Add Reference” to bring up the dialog. Then click the “Browse” tab, and browse to wherever Waveform Creator is installed. The default location is: C:\Program Files (x86)\Keysight\M9099 Waveform Creator

## Building a custom waveform plug-in (continued)

Select the DLL file called `Keysight.WaveformCreator.Core.Interfaces.dll`, as shown in Figure 3. Repeat the procedure to add a reference to the DLL file called `MathNet.Iridium.dll` which should be in the same directory as the interfaces DLL.

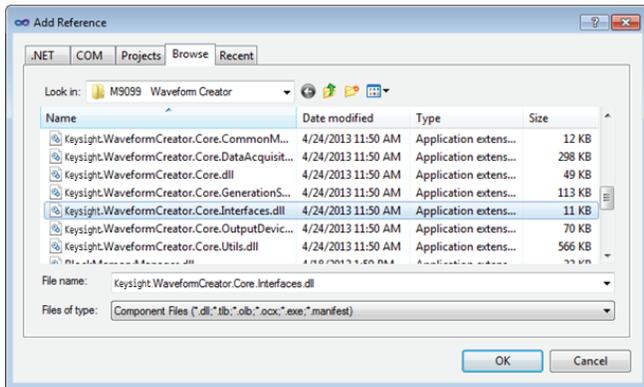


Figure 3. Add reference file browser window in Microsoft Visual Studio 2010

6. The Solution Explorer pane should now look like Figure 4. These are the only external references you will require to work with Waveform Creator.

- a. The `Keysight.WaveformCreator.Core.Interfaces` DLL defines the `IPlugInSegmentBuilder` interface.
- b. The `MathNet.Iridium` DLL defines the `Complex` class and provides other advanced math functions used by Waveform Creator. The `MathNet.Iridium` DLL is part of the open source Math.Net project “Iridium (Numerics)”. The latest version of this library is available from <http://www.mathdotnet.com/> and the documentation can be found at <http://www.mathdotnet.com/doc/Iridium.ashx>.

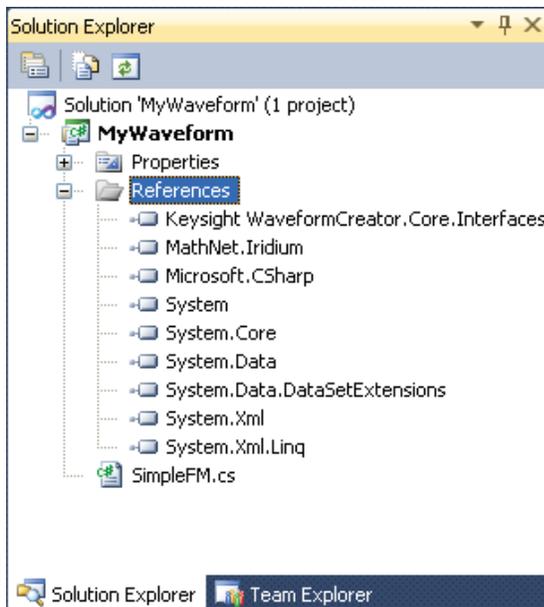


Figure 4. Solution explorer window with all references added.

## Building a custom waveform plug-in (continued)

7. Next, open or switch to the class definition code file and add the lines highlighted in bold font below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Keysight.WaveformCreator.Core.
    Interfaces;
using MathNet.Numerics;

namespace MyWaveform
{
    public class SimpleFM
    {
    }
}
```

8. Create a new class from the iterator class “IPlugInSegmentBuilder”.

```
namespace MyWaveform
{
    public class SimpleFM :
        IPlugInSegmentBuilder
    {
    }
}
```

9. Click on the word IPlugInSegmentBuilder, then hover over the small blue rectangle that appears under the letter I until a dropdown appears. Click and select “Implement interface ‘IPlugInSegmentBuilder’” as shown in Figure 5. This will auto generate a code template for your class with the necessary public methods and parameters shown below.

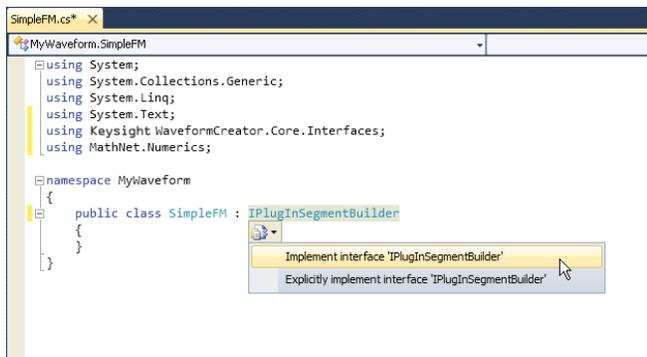


Figure 5. Selecting the implement interface option

## Building a custom waveform plug-in (continued)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Keysight.WaveformCreator.Core.
    Interfaces;
using MathNet.Numerics;

namespace MyWaveform
{
    public class SimpleFM :
        IPlugInSegmentBuilder
    {
        public double
        BuildWaveform(double targetSampleRate,
            ISegmentDataFileBuilder rawDataFile)
        {
            throw new
            NotImplementedException();
        }

        public string Name
        {
            get { throw new
            NotImplementedException(); }
        }
    }
}
```

As illustrated in the programming code at right:

10. Add the attribute [Serializable] to indicate that the class can be serialized (used for save / recall).
11. Re-work the definition of the Name method so that it can be read publicly but only set privately.
12. Add properties to control the modulation frequency, modulation index and the minimum number of samples in the output waveform.
13. Add a constructor method to initialize the various properties.
14. Add a return value to the main BuildWaveform method.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Keysight.WaveformCreator.Core.
    Interfaces;
using MathNet.Numerics;
namespace MyWaveform
{
    [Serializable]
    public class SimpleFM :
        IPlugInSegmentBuilder
    {
        public double Fm { get; set; }
        public double modIndex { get; set; }

        public int minSamples { get; set; }

        public SimpleFM()
        {
            Name = "Simple FM";
            Fm = 1000000.0;
            modIndex = 0.5;
            minSamples = 65536;
        }

        public double BuildWaveform(double
            targetSampleRate, ISegmentDataFileBuilder
            rawDataFile)
        {
            return 1.0 / targetSampleRate;
        }

        public string Name { get; private
            set; }
    }
}
```

At this point the code is buildable, but will not run in Waveform Creator because it will not write any data to "rawDataFile", which Waveform Creator would report as a zero-length segment. To continue, temporarily write an array of zero-valued complex numbers. Using MathNet.Numerics statement, simply add:

```
public double BuildWaveform(double
    targetSampleRate, ISegmentDataFileBuilder
    rawDataFile)
{
    Complex[] signal = new
    Complex[1000];
    rawDataFile.Write(signal);

    return 1.0 / targetSampleRate;
}
```

## Building a custom waveform plug-in (continued)

This completes the creation of a minimum viable waveform DLL. At this point, VS2010 will build the solution successfully with no errors or warnings. Since we are developing a plug-in for Waveform Creator we will need to run Waveform Creator from VS2010 to debug our code.

1. Configure VS2010 to copy the compiled plug-in code into Waveform Creator's plug-in directory:

- In the Solution Explorer pane, double click on the Properties folder to open the project properties
- Select the Build Events tab
- Select Edit Post-build and add the command:  

```
copy "$(TargetPath)" "C:\Program Files (x86)\Keysight\M9099 Waveform Creator\PlugIns\"
```

2. Make VS 2010 run Waveform Creator to start a debug session:

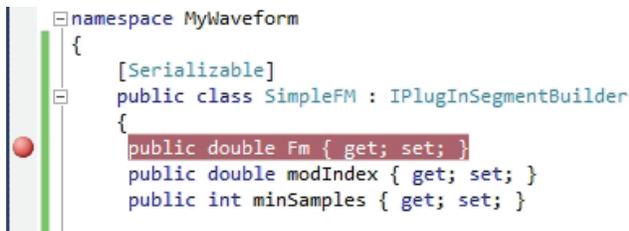
- Open the project properties (if it's not already open)
- Select the Debug tab
- Change the Start Action to the Start external program option and set the associated field to: C:\Program Files (x86)\Keysight\M9099 Waveform Creator\Keysight Waveform Creator.exe

Now you can start a debug run in VS2010.

- First, click Build (F6)
- Then, start Debugging (F5).

Waveform Creator should start up and you should see the "Simple FM" tab in the segment library. When you drag a segment into the Waveform Editor area, you should see the segment property grid appear on the left.

If the Simple FM tab does not appear in Waveform Creator, a breakpoint in the SimpleFM.cs file needs to be set to begin debugging. The code below illustrates where to place the breakpoint. Once set, execute the code in the debugger by pressing F5. Waveform Creator will load and execute the SimpleFM code and then stop at the breakpoint. You may press F10 to step over each line or Press F5 to continue executing the code.



```
namespace MyWaveform
{
    [Serializable]
    public class SimpleFM : IPlugInSegmentBuilder
    {
        public double Fm { get; set; }
        public double modIndex { get; set; }
        public int minSamples { get; set; }
    }
}
```

## Building a custom waveform plug-in (continued)

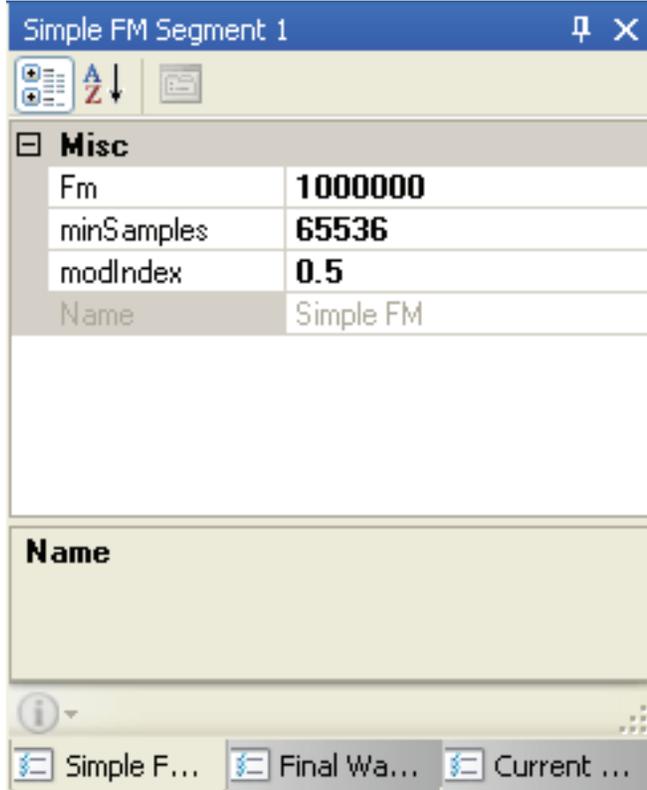


Figure 6. First pass property grid for Simple FM plug-in.

The appearance of the property grid is rather basic because it includes several defaults:

- All the properties have been placed in a “miscellaneous” category.
- The property name displayed in the list is set to the property’s internal variable name.
- The descriptive text in the help box at the bottom is set to the selected property’s internal variable name.
- Property grids show property values in bold when they are not equal to the specified default value. Since the default value for each of these properties has not yet been specified, zero is assumed to be the default value, and hence all the non-zero property values are bold.
- The name variable is shown as “read only” (greyed out) because only its get method is public.
- All of these can be adjusted by adding attributes to the class definition, as we illustrate below.

## Building a custom waveform plug-in (continued)

Once you have confirmed that you have functional plug-in code, exit Waveform Creator normally. This will, in turn, cause VS2010 to exit debugging so that you can continue with code development. The next step is to add attributes to the class definition as shown below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Keysight.WaveformCreator.Core.Interfaces;
using MathNet.Numerics;
using System.ComponentModel;

namespace MyWaveform
{
    public class SimpleFM : IPlugInSegmentBuilder
    {
        [DisplayName("Modulation Frequency (Hz)")]
        [Category("FM")]
        [Description("This sets the frequency of the sinusoidal single tone FM. Range is 0 to 1.0 GHz")]
        [DefaultValue(1000000.0)]
        public double Fm { get; set; }

        [DisplayName("Modulation index (unitless)")]
        [Category("FM")]
        [Description("This sets the modulation index of the FM. Range is 0 to 10.0")]
        [DefaultValue(0.5)]
        public double modIndex { get; set; }

        [DisplayName("Minimum samples (unitless)")]
        [Category("Waveform")]
        [Description("This sets a lower limit to the number of samples in the output waveform (useful when working with 89601B)")]
        [DefaultValue(65536)]
        public int minSamples { get; set; }

        public SimpleFM()
        {
            Name = "Simple FM";
            Fm = 1000000.0;
            modIndex = 0.5;
            minSamples = 65536;
        }

        public double BuildWaveform(double targetSampleRate, ISegmentDataFileBuilder rawDataFile)
        {
            Complex[] signal = new Complex[1000];
            rawDataFile.Write(signal);

            return 1.0 / targetSampleRate;
        }

        [Browsable(false)]
        public string Name { get; private set; }
    }
}
```

## Building a custom waveform plug-in (continued)

These property decorations have the following effects:

DisplayName	This allows you to define a more “friendly” description of the property, independent of the underlying property’s internal variable name.
Category	This allows you to gather the properties into named groups according to their purpose; mainly for usability reasons.
Description	This allows you to define context sensitive help text that will be displayed whenever the associated property is selected.
DefaultValue	This allows you to tell the PropertyGrid what value should be regarded as the default so that nondefault values can be highlighted in bold font.
	NOTE: This does not set the property variable to the default value; if non-zero property defaults are required, they must be explicitly set, either by a class constructor or after object instantiation. In this example we have already defined a class constructor to assign default values to the properties.
Browseable	This allows you to control whether a class property is shown in the property grid. In this example we have selected [Browseable(false)] for the Name property because the value of this property is more appropriately displayed as a label on the segment library tab.

When we build the decorated class and re-run Waveform Center, we can see the effect of these changes on the segment property grid.

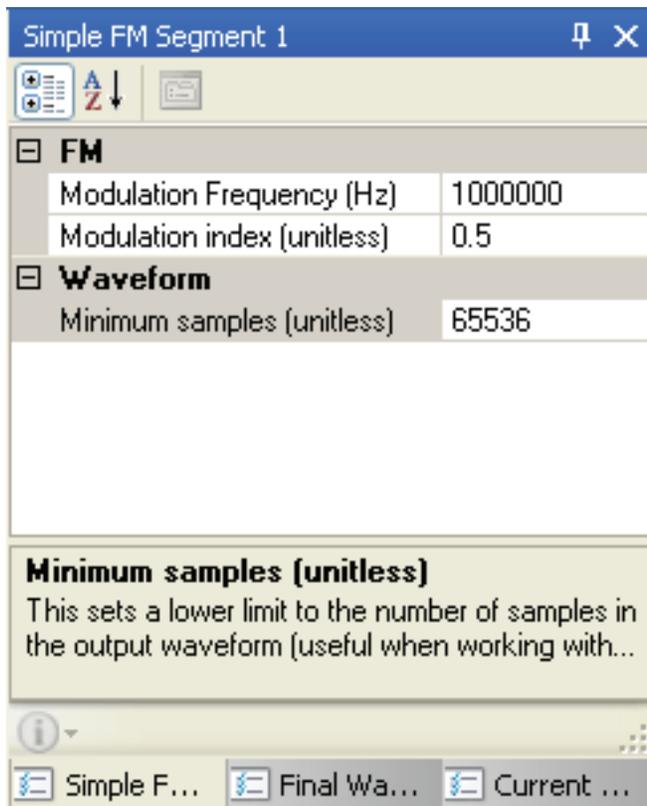


Figure 7. Final version of property grid for Simple FM plug-in.

## Building a custom waveform plug-in (continued)

Unlike Waveform Center's native property grids which can invoke Keysight proprietary units conversion code, the user defined plug-in's property grid can only handle basic data types such as int, double, string etc. To keep this tutorial example focused on the essentials we have chosen to make the property units implicit, but you may wish to implement more sophisticated units parsing based on string parameters. At this point the framework programming is complete; all that remains is to implement the FM signal generation code, so exit Waveform Center normally to exit debugging and return to source development, then add the FM code.

## Summary

Waveform Creator provides a simple, yet flexible environment to create customized digitally modulated signals for radar, military radios and other similar applications. The ability to create your own custom waveform plug-ins, as illustrated in this application note, greatly enhances Waveform Creator's capabilities beyond those provided with the core software. By integrating your own custom waveforms with Waveform Creator's core features, the types of signals you can create are limited only by your imagination.

For your reference, the programming code is shown below with new lines emphasized in bold. The code can be downloaded from the Knowledge Center at <http://edocs.soco.keysight.com/x/PQsTDQ>

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Keysight.WaveformCreator.Core.Interfaces;
using MathNet.Numerics;
using System.ComponentModel;
namespace MyWaveform
{
    public class SimpleFM : IPlugInSegmentBuilder
    {
        [DisplayName("Modulation Frequency (Hz)")]
        [Category("FM")]
        [Description("This sets the frequency of the sinusoidal single tone FM. Range is 0 to 1.0
GHz")]
        [DefaultValue(1000000.0)]
        public double Fm { get; set; }

        [DisplayName("Modulation index (unitless)")]
        [Category("FM")]
        [Description("This sets the modulation index of the FM. Range is 0 to 10.0")]
        [DefaultValue(0.5)]
        public double modIndex { get; set; }

        [DisplayName("Minimum samples (unitless)")]
        [Category("Waveform")]
        [Description("This sets a lower limit to the number of samples in the output waveform
(useful when working with 89601B)")]
        [DefaultValue(65536)]
        public int minSamples { get; set; }

        public SimpleFM()
        {
            Name = "Simple FM";
            Fm = 1000000.0;
            modIndex = 0.5;
            minSamples = 65536;
        }

        public double BuildWaveform(double targetSampleRate, ISegmentDataFileBuilder rawDataFile)
        {
            int samplesPerCycle;
            int nSamples;
            int i;
            double time;
            double angle;
            double Fs;

            samplesPerCycle = (int)(targetSampleRate / Fm);

            if (samplesPerCycle > 100) samplesPerCycle = 100;
            Fs = samplesPerCycle * Fm;

            nSamples = samplesPerCycle * (int)Math.Ceiling((decimal)minSamples / (decimal)
samplesPerCycle);

            Complex[] signal = new Complex[nSamples];

            for (i = 0; i < nSamples; i++)
            {
                time = i * (1.0 / Fs);
                angle = modIndex * Math.Sin(2.0 * Math.PI * Fm * time);
                signal[i] = Complex.FromRealImaginary(Math.Cos(angle), Math.Sin(angle));
            }

            rawDataFile.Write(signal);

            return 1.0 / Fs;
        }

        [Browsable(false)]
        public string Name { get; private set; }
    }
}

```

**myKeysight**

**myKeysight**

[www.keysight.com/find/mykeysight](http://www.keysight.com/find/mykeysight)

A personalized view into the information most relevant to you.



[www.axiestandard.org](http://www.axiestandard.org)

AdvancedTCA® Extensions for Instrumentation and Test (AXIe) is an open standard that extends the AdvancedTCA for general purpose and semiconductor test. Keysight is a founding member of the AXIe consortium. ATCA®, AdvancedTCA®, and the ATCA logo are registered US trademarks of the PCI Industrial Computer Manufacturers Group.



[www.lxistandard.org](http://www.lxistandard.org)

LAN eXtensions for Instruments puts the power of Ethernet and the Web inside your test systems. Keysight is a founding member of the LXI consortium.



[www.pxisa.org](http://www.pxisa.org)

PCI eXtensions for Instrumentation (PXI) modular instrumentation delivers a rugged, PC-based high-performance measurement and automation system.



**Three-Year Warranty**

[www.keysight.com/find/ThreeYearWarranty](http://www.keysight.com/find/ThreeYearWarranty)

Keysight's commitment to superior product quality and lower total cost of ownership. The only test and measurement company with three-year warranty standard on all instruments, worldwide.



**Keysight Assurance Plans**

[www.keysight.com/find/AssurancePlans](http://www.keysight.com/find/AssurancePlans)

Up to five years of protection and no budgetary surprises to ensure your instruments are operating to specification so you can rely on accurate measurements.



[www.keysight.com/quality](http://www.keysight.com/quality)

Keysight Technologies, Inc.  
DEKRA Certified ISO 9001:2008  
Quality Management System

**Keysight Channel Partners**

[www.keysight.com/find/channelpartners](http://www.keysight.com/find/channelpartners)

Get the best of both worlds: Keysight's measurement expertise and product breadth, combined with channel partner convenience.

[www.keysight.com/find/modular](http://www.keysight.com/find/modular)

[www.keysight.com/find/m9099](http://www.keysight.com/find/m9099)

For more information on Keysight Technologies' products, applications or services, please contact your local Keysight office. The complete list is available at: [www.keysight.com/find/contactus](http://www.keysight.com/find/contactus)

**Americas**

Canada	(877) 894 4414
Brazil	55 11 3351 7010
Mexico	001 800 254 2440
United States	(800) 829 4444

**Asia Pacific**

Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Other AP Countries	(65) 6375 8100

**Europe & Middle East**

Austria	0800 001122
Belgium	0800 58580
Finland	0800 523252
France	0805 980333
Germany	0800 6270999
Ireland	1800 832700
Israel	1 809 343051
Italy	800 599100
Luxembourg	+32 800 58580
Netherlands	0800 0233200
Russia	8800 5009286
Spain	0800 000154
Sweden	0200 882255
Switzerland	0800 805353
	Opt. 1 (DE)
	Opt. 2 (FR)
	Opt. 3 (IT)
United Kingdom	0800 0260637

For other unlisted countries:  
[www.keysight.com/find/contactus](http://www.keysight.com/find/contactus)  
(BP-07-10-14)

