pyarbtools Documentation

Release 2023.06.1

Morgan Allison

Jun 14, 2023

CONTENTS:

CHAPTER

ONE

PYARBTOOLS: KEYSIGHT SIGNAL GENERATOR CONTROL & WAVEFORM CREATION

Current version: 2023.06.1

Frustrated after looking through hundreds of pages of user manuals to find out how to download a waveform to a signal generator?

Need to test your amplifier or filter with a complex signal but don't want to crack open your digital signal processing books from college?

Tired of troubleshooting VISA connections, conflicts, and incompatibilities?

Can't get a Matlab license or the correct toolbox(es) for your work?

Try PyArbTools: a fast, free, and flexible way to create waveforms and control Keysight signal generators.

PyArbTools is a collection of Python classes and functions that provide basic signal creation, instrument configuration, and waveform download capabilities for Keysight signal sources.

It is *loosely* based on Keysight's IQ Tools, a Matlab-based toolkit that accomplishes similar things. PyArbTools was built to satisfy the needs of signal generator users who can't/don't want to use Matlab and to improve code readability and documentation.

Features

- **New in 2023.06.1:** Choose between direct socket communication and PyVISA for all instruments.
- Supports M8190A, M8195A, and M8196A arbitrary waveform generators, N5182B MXG/N5172B EXG/E8267D PSG vector signal generators, and the M9384B VXG vector signal generator.
- Connect to and configure instruments, download waveforms, and control playback, all using easy-to-use functions rather than a list of SCPI commands.
- Create sequences on the M8190A.
- Calibrate waveforms using Keysight's 89600 VSA software.
- For custom applications, communicate with instruments using SCPI commands.

DOCUMENTATION

Take a look at pyarbtools/examples.py for sample code.

PyArbTools was written for Python and is not currently compatible with legacy Python 2.x

License: GPL 3

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install pyarbtools, run this command in your terminal:

```
$ pip install pyarbtools
```

This is the preferred method to install pyarbtools, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for pyarbtools can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/morgan-at-keysight/pyarbtools
```

Or download the tarball:

```
$ curl  -OL https://github.com/morgan-at-keysight/pyarbtools/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use PyArbTools in a project:

```
import pyarbtools
```

PyArbTools is built from a few primary submodules:

- *instruments*
- *wfmBuilder*
- *vsaControl*
- *GUI*

Supported instruments include:

- *M8190A* AWG
- *M8195A* AWG
- *M8196A* AWG
- ***VSG***
  - E8267D PSG
  - N5182B MXG
  - N5172B EXG
  - M9381A/M9383A
- *VXG*

Supported waveform building functions include:

- *export_wfm*
- *import_mat*
- *zero_generator*
- *sine_generator*
- *am_generator*
- *cw_pulse_generator*
- *chirp_generator*
- *barker_generator*
- *multitone_generator*

- *digmod_generator*

Supported VSA control functions include:

- *acquire_continuous*
- *acquire_single*
- *stop*
- *autorange*
- *set_hw*
- *set_cf*
- *set_span*
- *set_measurement*
- *configure_ddemod*
- *configure_vector*
- *recall_recording*
- *sanity_check*

3.1 instruments

To use/control a signal generator, create a class of the signal generator's instrument type and enter the instrument's IP address as the first argument. There are additional keyword arguments you can add to set things like `apiType`, `protocol`, `port`, `timeout`, and `reset`:

```
# Example
awg = pyarbtools.instruments.M8910A('192.168.1.12')
vsg = pyarbtools.instruments.VSG('192.168.1.13', port=5025, timeout=10, reset=True)
vxg = pyarbtools.instruments.VXG('192.168.1.14', apiType='pyvisa', protocol='hislip',␣
↪port=1)
```

Every class is built on either `'socketscpi'` or `'pyvisa'` and allows the user to send SCPI commands/queries, send/receive data using IEEE 488.2 binary block format, check for errors, and gracefully disconnect from the instrument. This architectural decision to include an open SCPI interface was made to provide additional flexibility for users who need to use specific setup commands *not* covered by built-in functions:

```
# Example
awg.write('*RST')
instID = awg.query('*IDN?')
awg.write_binary_values('trace:data 1, 0, ', data, datatype='h')
awg.disconnect()
```

As of version 2023.06.0, when an instance of an instrument is created, users can choose to use either `'socketscpi'` (a lightweight and fast raw socket connection) or `'pyvisa'` (full-featured industry standard) as the underlying instrument communications platform. This can be done using the `apiType` keyword argument. If `'pyvisa'` is chosen, the `protocol` and `port` must also be chosen. If `'socketscpi'` is chosen, only port must be specified. When an instrument class is created, PyArbTools connects to the instrument at the IP address given by the user and sends a few queries to populate some class attributes.

Each class constructor has a `reset` keyword argument that causes the instrument to perform a default setup prior to running the rest of the code. It's set to `False` by default to prevent unwanted settings changes.

Each instrument class includes a `.download_wfm()` method, which takes care of the instrument's binary formatting, minimum length, and granularity requirements for you. It also makes a reasonable effort to correct for length/granularity violations and raises a descriptive exception if any requirements aren't met by the waveform:

```
# Example
iq = pyarbtools.wfmBuilder.multitone_generator(fs=100e6, spacing=1e6, num=11, wfmFormat=
↪'iq')
vsg.download_wfm(iq)

real = pyarbtools.wfmBuilder.cw_pulse_generator(fs=12e9, spacing=1e6, num=11, cf=1e9,␣
↪wfmFormat='real')
awg.download_wfm(real)
```

Each instrument class also includes a `.configure()` method. It provides keyword arguments to configure selected settings on the signal generator *and sets relevant class attributes* so that the user knows how the generator is configured and can use those variables in code without having to send a SCPI query to determine values:

```
awg.configure(res='wsp', clkSrc='int', fs=7.2e9)
print(f'Sample rate is {awg.fs} samples/sec.')
print(f'Clock source is {awg.clkSrc}.')

recordLength = 1000
print(f'Waveform play time is {recordLength / awg.fs} seconds.')
```

3.2 M8190A

```
awg = pyarbtools.instruments.M8190A(ipAddress, apiType='socketspi', port=5025,␣
↪timeout=10, reset=False)
awg = pyarbtools.instruments.M8190A(ipAddress, apiType='pyvisa', protocol='hislip',␣
↪port=0, timeout=10, reset=False)
```

Arguments

- `ipAddress (str)`: IP address of the instrument.

Keyword Arguments

- `apiType (str)`: API type to use. `'socketscpi'` or `'pyvisa'`, default is `'socketscpi'`.
- `protocol (str)`: Only used when `apiType='pyvisa'`. Chooses which VISA protocol to use. `'hislip'` or `'vxi11'`.
- `port (int)`: Selects socket port when `apiType='socketscpi'`. Selects protocol port when `apiType='pyvisa'`.
- `timeout (int)`: Sets the instrument timeout in seconds. Default is `10`.
- `reset (bool)`: Determines if the instrument will be preset when object is created. `True` or `False`, default is `False`

3.2.1 attributes

These attributes are automatically populated when connecting to the instrument and when calling the `.configure()` method. Generally speaking, they are also the keyword arguments for `.configure()`.

- `instId (str)`: Instrument identifier. Contains instrument model, serial number, and firmware revision.
- `res (str)`: AWG resolution. Values are `'wpr'` (14 bit), `'wsp'` (12 bit) (default), `'intx3'`, `'intx12'`, `'intx24'`, or `'intx48'` (intxX resolutions are all 15 bit).
- `clkSrc (str)`: Sample clock source. Values are `'int'` (default) or `'ext'`.
- `fs (float)`: Sample rate in Hz. Values range from `125e6` to `12e9`. Default is `7.2e9`.
- `refSrc (str)`: Reference clock source. Values are `'axi'` (default), `'int'`, `'ext'`.
- `refFreq (float)`: Reference clock frequency in Hz. Values range from `1e6` to `200e6` in steps of `1e6`. Default is `100e6`.
- `out1`, `out2 (str)`: Output signal path for channel 1 and 2 respectively. Values are `'dac'` (default), `'dc'`, `'ac'`.
- `amp1`, `amp2 (float)`: Output amplitude for channel 1 and 2 respectively. Values depend on output path chosen.
- `func1`, `func2 (str)`: Function of channel 1 and 2 respectively. Values are `'arb'` (default), `'sts'` (sequence), or `'stc'` (scenario).
- `cf1`, `cf2 (str)`: Carrier frequency in Hz of channel 1 and 2 respectively. This setting is only applicable if the digital upconverter is being used (`res` arguments of `'intx<#>'`). Value range is `0` to `12e9`.

```
print(f'AWG Clock Source: {awg.clkSrc}.')
>>> AWG Clock Source: int.
```

3.2.2 configure

```
M8190A.configure(**kwargs)
# Example
M8190A.configure(fs=12e9, out1='dac', func1='arb')
```

Sets the basic configuration for the M8190A and populates class attributes accordingly. It *only* changes the setting(s) for the keyword argument(s) sent by the user.

Keyword Arguments

- `res (str)`: AWG resolution. Arguments are `'wpr'` (14 bit), `'wsp'` (12 bit) (default), `'intx3'`, `'intx12'`, `'intx24'`, or `'intx48'` (intxX resolutions are all 15 bit).
- `clkSrc (str)`: Sample clock source. Arguments are `'int'` (default) or `'ext'`.
- `fs (float)`: Sample rate in Hz. Argument range is `125e6` to `12e9`. Default is `7.2e9`.
- `refSrc (str)`: Reference clock source. Arguments are `'axi'` (default), `'int'`, `'ext'`.
- `refFreq (float)`: Reference clock frequency in Hz. Argument range is `1e6` to `200e6` in steps of `1e6`. Default is `100e6`.
- `out1`, `out2 (str)`: Output signal path for channel 1 and 2 respectively. Arguments are `'dac'` (default), `'dc'`, `'ac'`.
- `amp1`, `amp2 (float)`: Output amplitude for channel 1 and 2 respectively. Argument range varies depending on output path chosen.

- `func1`, `func2 (str)`: Function of channel 1 and 2 respectively. Arguments are `'arb'` (default), `'sts'` (sequence), or `'stc'` (scenario).
- `cf1`, `cf2 (str)`: Carrier frequency in Hz of channel 1 and 2 respectively. This setting is only applicable if the digital upconverter is being used (`res` arguments of `'intx<#>'`). Argument range is `0` to `12e9`.

Returns

- None

3.2.3 download_wfm

```
M8190A.download_wfm(wfmData, ch=1, name='wfm', wfmFormat='iq', sampleMkr=0,␣
↪sampleMkrLength=240, syncMkr=0, syncMkrLength=240)
```

Defines and downloads a waveform into the lowest available segment slot.

Arguments

- `wfmData (NumPy array)`: Array of waveform samples (either real or IQ).
- `ch (int)`: Channel to which waveform will be assigned. Arguments are `1` (default) or `2`.
- `name (str)`: Name for downloaded waveform segment.
- `wfmFormat (str)`: Format of the waveform being downloaded. Arguments are `'iq'` (default) or `'real'`.
- `sampleMkr (int)`: Index of the beginning of the sample marker.
- `sampleMkrLength (int)`: Length in samples of the sample marker. Default is 240.
- `syncMkr (int)`: Index of the beginning of the sync marker. Currently, marker width is 240 samples.
- `syncMkrLength (int)`: Length in samples of the sync marker. Default is 240.

Returns

- `(int)`: Segment identifier used to specify which waveform is played using `.play()`.

3.2.4 delete_segment

```
M8190A.delete_segment(wfmID=1, ch=1)
```

Deletes a waveform segment from the waveform memory.

Arguments

- `wfmID (int)`: Segment number used to specify which waveform is deleted.
- `ch (int)`: Channel from which waveform will be deleted. Arguments are `1` (default) or `2`.

Returns

- None

3.2.5 clear_all_wfm

```
M8190A.clear_all_wfm()
```

Stops playback and deletes all waveform segments from the waveform memory.

Arguments

- None

Returns

- None

3.2.6 play

```
M8190A.play(wfmID=1, ch=1)
```

Selects waveform, turns on analog output, and begins continuous playback.

Arguments

- `wfmID (int)`: Waveform identifier, used to select waveform to be played. Default is 1.
- `ch (int)`: Channel to be used for playback. Default is 1.

Returns

- None

3.2.7 stop

```
M8190A.stop(ch=1)
```

Turns off analog output and stops playback.

Arguments

- `ch (int)`: Channel to be stopped. Default is 1.

Returns

- None

3.2.8 create_sequence

```
M8190A.create_sequence(numSteps, ch=1)
```

Deletes all sequences and creates a new sequence.

Arguments

- `numSteps (int)`: Number of steps in the sequence. Max is 512k.
- `ch (int)`: Channel for which the sequence is created. Values are 1 or 2. Default is 1.

Returns

- None

3.2.9 insert_wfm_in_sequence

```
M8190A.insert_wfm_in_sequence(wfmID, seqIndex, seqStart=False, seqEnd=False,␣
↪markerEnable=False, segAdvance='auto', loopCount=1, startOffset=1,␣
↪endOffset=0xFFFFFFFF, ch=1)
```

Inserts a specific waveform segment into a specific index in the sequence.

Arguments

- `wfmID (int)`: Identifier/number of the segment to be added to the sequence. Argument should be taken from the return value of `download_wfm()`.
- `seqIndex (int)`: Index in the sequence where the segment should be added. Argument range is `0` to `numSteps - 1`.
- `seqStart (bool)`: Determines if this segment is the start of the sequence.
- `seqEnd (bool)`: Determines if this segment is the end of the sequence.
- `markerEnable (bool)`: Enables or disables the marker for this segment.
- `segAdvance (str)`: Defines segment advance behavior. `'auto'`, `'conditional'`, `'repeat'`, `'single'`. Default is `'auto'`.
- `loopCount (int)`: Determines how many times this segment will be repeated. Argument range is `1` to `4294967295`.
- `startOffset (int)`: Determines the start offset of the waveform in samples if only a part of the waveform is to be used. Default is `0` and should likely remain that way.
- `endOffset (int)`: Determines the end offset of the waveform in samples if only a part of the waveform is to be used. Default is the hex value `0xffffffff` and should likely remain that way. Note that `endOffset` is zero-indexed, so if you want an offset of 1000, use 999.
- `ch (int)`: Channel for which the sequence is created. Values are `1` or `2`. Default is `1`.

Returns

- None

3.2.10 insert_idle_in_sequence

```
M8190A.insert_idle_in_sequence(seqIndex, seqStart=False, idleSample=0, idleDelay=640,␣
↪ch=1)
```

Inserts an idle segment into a specific index in the sequence.

Arguments

- `seqIndex (int)`: Index in the sequence where the segment should be added. Argument range is `0` to `numSteps - 1`.
- `seqStart (bool)`: Determines if this segment is the start of the sequence.
- `idleSample (float)`: Sample value to be used as the DAC output during idle time. Default is `0`.
- `idleDelay (int)`: Duration of the idle segment in samples. Argument range is `10 * granularity` to `(2**25 * granularity) + (granularity - 1)` Default is `640`.
- `ch (int)`: Channel for which the sequence is created. Values are `1` or `2`. Default is `1`.

Returns

- None

3.3 M8195A

```
awg = pyarbtools.instruments.M8195A(ipAddress, port=5025, timeout=10, reset=False)
awg = pyarbtools.instruments.M8195A(ipAddress, apiType='pyvisa', protocol='hislip',␣
↪port=0, timeout=10, reset=False)
```

Arguments

- `ipAddress (str)`: IP address of the instrument.

Keyword Arguments

- `apiType (str)`: API type to use. `'socketscpi'` or `'pyvisa'`, default is `'socketscpi'`.
- `protocol (str)`: Only used when `apiType='pyvisa'`. Chooses which VISA protocol to use. `'hislip'` or `'vxi11'`.
- `port (int)`: Selects socket port when `apiType='socketscpi'`. Selects protocol port when `apiType='pyvisa'`.
- `timeout (int)`: Sets the instrument timeout in seconds. Default is `10`.
- `reset (bool)`: Determines if the instrument will be preset when object is created. `True` or `False`, default is `False`

3.3.1 attributes

These attributes are automatically populated when connecting to the instrument and when calling the `.configure()` method. Generally speaking, they are also the keyword arguments for `.configure()`.

- `instId (str)`: Instrument identifier. Contains instrument model, serial number, and firmware revision.
- `dacMode (str)`: Sets the DAC mode. Values are `'single'` (default), `'dual'`, `'four'`, `'marker'`, `'dcd'`, or `'dcm'`.
- `memDiv (str)`: Clock/memory divider rate. Values are `1`, `2`, or `4`.
- `fs (float)`: Sample rate in Hz. Values range from `53.76e9` to `65e9`.
- `refSrc (str)`: Reference clock source. Values are `'axi'` (default), `'int'`, `'ext'`.
- `refFreq (float)`: Reference clock frequency in Hz. Values range from `10e6` to `300e6` in steps of `1e6`. Default is `100e6`.
- `amp1/2/3/4 (float)`: Output amplitude for a given channel in volts pk-pk. (min=75 mV, max=1 V)
- `func (str)`: Function of channels. Values are `'arb'` (default), `'sts'`, or `'stc'`.

```
print(f'AWG Channel 1 Amplitude: {awg.amp1} Vpp.')
>>> AWG Channel 1 Amplitude: 0.750 Vpp.
```

3.3.2 configure

```
M8195A.configure(**kwargs)
# Example
M8195A.configure(dacMode='single', fs=64e9)
```

Sets the basic configuration for the M8195A and populates class attributes accordingly. It *only* changes the setting(s) for the keyword argument(s) sent by the user.

Arguments

- `dacMode (str)`: Sets the DAC mode. Arguments are `'single'` (default), `'dual'`, `'four'`, `'marker'`, `'dcd'`, or `'dcm'`.
- `memDiv (str)`: Clock/memory divider rate. Arguments are `1`, `2`, or `4`.
- `fs (float)`: Sample rate in Hz. Argument range is `53.76e9` to `65e9`.
- `refSrc (str)`: Reference clock source. Arguments are `'axi'` (default), `'int'`, `'ext'`.
- `refFreq (float)`: Reference clock frequency in Hz. Argument range is `10e6` to `300e6` in steps of `1e6`. Default is `100e6`.
- `amp1/2/3/4 (float)`: Output amplitude for a given channel in volts pk-pk. (min=75 mV, max=1 V)
- `func (str)`: Function of channels. Arguments are `'arb'` (default), `'sts'`, or `'stc'`.

Returns

- None

3.3.3 download_wfm

```
M8195A.download_wfm(wfmData, ch=1, name='wfm')
```

Defines and downloads a waveform into the lowest available segment slot. Returns useful waveform identifier.

Arguments

- `wfmData (NumPy array)`: Array containing real waveform samples (not IQ).
- `ch (int)`: Channel to which waveform will be assigned. Arguments are `1` (default), `2`, `3`, or `4`.
- `name (str)`: String providing a name for downloaded waveform segment.

Returns

- `(int)`: Segment number used to specify which waveform is played using `.play()`.

3.3.4 delete_segment

```
M8195A.delete_segment(wfmID=1, ch=1)
```

Deletes a waveform segment from the waveform memory.

Arguments

- `wfmID (int)`: Segment number used to specify which waveform is deleted.
- `ch (int)`: Channel from which waveform will be deleted. Arguments are `1` (default), `2`, `3`, `4`.

Returns

- None

3.3.5 clear_all_wfm

```
M8195A.clear_all_wfm()
```

Stops playback and deletes all waveform segments from the waveform memory.

Arguments

- None

Returns

- None

3.3.6 play

```
M8195A.play(wfmID=1, ch=1)
```

Selects waveform, turns on analog output, and begins continuous playback.

Arguments

- `wfmID (int)`: Segment index of the waveform to be loaded. Default is 1.
- `ch (int)`: Channel to be used for playback. Arguments are 1 (default), 2, 3, 4.

Returns

- None

3.3.7 stop

```
M8195A.stop(ch=1)
```

Turns off analog output and stops playback.

Arguments

- `ch (int)`: Channel to be stopped. Default is 1.

Returns

- None

3.4 M8196A

```
awg = pyarbtools.instruments.M8196A(ipAddress port=5025, timeout=10, reset=False)
awg = pyarbtools.instruments.M8196A(ipAddress, apiType='pyvisa', protocol='hislip',␣
↪port=0, timeout=10, reset=False)
```

Arguments

- `ipAddress (str)`: IP address of the instrument.

Keyword Arguments

- `apiType (str)`: API type to use. `'socketscpi'` or `'pyvisa'`, default is `'socketscpi'`.
- `protocol (str)`: Only used when `apiType='pyvisa'`. Chooses which VISA protocol to use. `'hislip'` or `'vxi11'`.
- `port (int)`: Selects socket port when `apiType='socketscpi'`. Selects protocol port when `apiType='pyvisa'`.
- `timeout (int)`: Sets the instrument timeout in seconds. Default is `10`.
- `reset (bool)`: Determines if the instrument will be preset when object is created. `True` or `False`, default is `False`

3.4.1 attributes

These attributes are automatically populated when connecting to the instrument and when calling the `.configure()` method. Generally speaking, they are also the keyword arguments for `.configure()`.

- `instId (str)`: Instrument identifier. Contains instrument model, serial number, and firmware revision.
- `dacMode (str)`: Sets the DAC mode. Values are `'single'` (default), `'dual'`, `'four'`, `'marker'`, or `'dcmarker'`.
- `fs (float)`: Sample rate. Values range from `82.24e9` to `93.4e9`.
- `refSrc (str)`: Reference clock source. Values are `'axi'` (default), `'int'`, `'ext'`.
- `refFreq (float)`: Reference clock frequency. Values range from `10e6` to `17e9`. Default is `100e6`.

```
print(f'AWG DAC Mode: {awg.dacMode}.')
>>> AWG DAC Mode: SINGLE.
```

3.4.2 configure

```
M8196A.configure(**kwargs)
# Example
M8196A.configure(dacMode='single', fs=92e9)
```

Sets the basic configuration for the M8196A and populates class attributes accordingly. It *only* changes the setting(s) for the keyword argument(s) sent by the user.

Arguments

- `dacMode (str)`: Sets the DAC mode. Arguments are `'single'` (default), `'dual'`, `'four'`, `'marker'`, or `'dcmarker'`.

- `fs (float)`: Sample rate. Argument range is `82.24e9` to `93.4e9`.
- `refSrc (str)`: Reference clock source. Arguments are `'axi'` (default), `'int'`, `'ext'`.
- `refFreq (float)`: Reference clock frequency. Argument range is `10e6` to `17e9`. Default is `100e6`.

Returns

- None

3.4.3 download_wfm

```
M8196A.download_wfm(wfmData, ch=1, name='wfm')
```

Defines and downloads a waveform into the lowest available segment slot. Returns useful waveform identifier.

Arguments

- `wfmData (NumPy array)`: Array containing real waveform samples (not IQ).
- `ch (int)`: Channel to which waveform will be assigned. Arguments are 1 (default), 2, 3, or 4.
- `name (str)`: Name for downloaded waveform segment.

Returns

- `(int)`: Segment number used to specify which waveform is played using `.play()`.

3.4.4 delete_segment

```
M8196A.delete_segment(wfmID=1, ch=1)
```

Deletes a waveform segment from the waveform memory.

Arguments

- `wfmID (int)`: Segment number used to specify which waveform is deleted.
- `ch (int)`: Channel from which waveform will be deleted. Arguments are 1 (default), 2, 3, 4.

Returns

- None

3.4.5 clear_all_wfm

```
M8196A.clear_all_wfm()
```

Stops playback and deletes all waveform segments from the waveform memory.

Arguments

- None

Returns

- None

3.4.6 play

```
M8196A.play(ch=1)
```

Selects waveform, turns on analog output, and begins continuous playback.

Arguments

- `ch (int)`: Channel to be used for playback. Arguments are `1` (default), `2`, `3`, `4`.

Returns

- None

3.4.7 stop

```
M8196A.stop(ch=1)
```

Turns off analog output and stops playback.

Arguments

- `ch (int)`: Channel to be stopped. Default is `1`.

Returns

- None

3.5 VSG

```
vsg = pyarbtools.instruments.VSG(ipAddress port=5025, timeout=10, reset=False)
vsg = pyarbtools.instruments.VXG(ipAddress, apiType='pyvisa', protocol='hislip', port=0,␣
↪timeout=10, reset=False)
```

Arguments

- `ipAddress (str)`: IP address of the instrument.

Keyword Arguments

- `apiType (str)`: API type to use. `'socketscpi'` or `'pyvisa'`, default is `'socketscpi'`.
- `protocol (str)`: Only used when `apiType='pyvisa'`. Chooses which VISA protocol to use. `'hislip'` or `'vxi11'`.
- `port (int)`: Selects socket port when `apiType='socketscpi'`. Selects protocol port when `apiType='pyvisa'`.
- `timeout (int)`: Sets the instrument timeout in seconds. Default is `10`.
- `reset (bool)`: Determines if the instrument will be preset when object is created. `True` or `False`, default is `False`

3.5.1 attributes

These attributes are automatically populated when connecting to the instrument and when calling the `.configure()` method. Generally speaking, they are also the keyword arguments for `.configure()`.

- `instId (str)`: Instrument identifier. Contains instrument model, serial number, and firmware revision.
- `rfState (int)`: RF output state. Values are `0` (default) or `1`.
- `modState (int)`: Modulation state. Values are `0` (default) or `1`.
- `arbState (int)`: Internal arb state. Values are `0` (default) or `1`.
- **`cf (float)`: Output carrier frequency in Hz. Value range is instrument dependent. Default is `1e9`.**
  - EXG/MXG: `9e3` to `6e9`
  - PSG: `100e3` to `44e9`
- **`amp (float)`: Output power in dBm. Value range is instrument dependent. Default is `-130`.**
  - EXG/MXG: `-144` to `+26`
  - PSG: `-130` to `+21`
- `alcState (int)`: ALC (automatic level control) state. Values are `1` or `0` (default).
- `iqScale (int)`: IQ scale factor in %. Values range from `1` to `100`. Default is `70`.
- `refSrc (str)`: Reference clock source. Values are `'int'` (default), or `'ext'`.
- **`fs (float)`: Sample rate in Hz. Values range is instrument dependent.**
  - EXG/MXG: `1e3` to `200e6`
  - PSG: `1` to `100e6`

```
print(f'VSG Sample Rate: {vsg.fs} samples/sec.')
>>> VSG Sample Rate: 200000000 samples/sec.
```

3.5.2 configure

```
VSG.configure(**kwargs)
# Example
VSG.configure(rfState=1, cf=1e9, amp=-20)
```

Sets the basic configuration for the VSG and populates class attributes accordingly. It *only* changes the setting(s) for the keyword argument(s) sent by the user.

Arguments

- `rfState (int)`: Turns the RF output state on or off. Arguments are `0` (default) or `1`.
- `modState (int)`: Turns the modulation state on or off. Arguments are `0` (default) or `1`.
- `arbState (int)`: Turns the internal arb on or off. Arguments are `0` (default) or `1`.
- **`cf (float)`: Output carrier frequency in Hz. Argument range is instrument dependent. Default is `1e9`.**
  - EXG/MXG: `9e3` to `6e9`
  - PSG: `100e3` to `44e9`
- **`amp (float)`: Output power in dBm. Argument range is instrument dependent. Default is `-130`.**

  - EXG/MXG: -144 to +26
  - PSG: -130 to +21
- alcState (int): Turns the ALC (automatic level control) on or off. Arguments are 1 or 0 (default).
- iqScale (int): IQ scale factor in %. Argument range is 1 to 100. Default is 70.
- refSrc (str): Reference clock source. Arguments are 'int' (default), or 'ext'.
- **fs (float): Sample rate in Hz. Argument range is instrument dependent.**
  - EXG/MXG: 1e3 to 200e6
  - PSG: 1 to 100e6

Returns

- None

3.5.3 download_wfm

```
VSG.download_wfm(wfmData, wfmID='wfm')
```

Defines and downloads a waveform into WFM1: memory directory and checks that the waveform meets minimum waveform length and granularity requirements. Returns useful waveform identifier.

Arguments

- wfmData (NumPy array): Array of values containing the complex sample pairs in an IQ waveform.
- wfmID (str): Name of the waveform to be downloaded. Default is 'wfm'.

Returns

- wfmID (string): Useful waveform name or identifier. Use this as the waveform identifier for .play().

3.5.4 delete_wfm

```
VSG.delete_wfm(wfmID)
```

Deletes a waveform from the waveform memory.

Arguments

- wfmID (str): Name of the waveform to be deleted.

Returns

- None

3.5.5 clear_all_wfm

```
VSG.clear_all_wfm()
```

Stops playback and deletes all waveforms from the waveform memory.

Arguments

- None

Returns

- None

3.5.6 play

```
VSG.play(wfmID='wfm')
```

Selects waveform and activates arb mode, RF output, and modulation.

Arguments

- `wfmID (str)`: Name of the waveform to be loaded. Default is `'wfm'`.

Returns

- None

3.5.7 stop

```
VSG.stop()
```

Deactivates arb mode, RF output, and modulation.

Arguments

- None

Returns

- None

3.6 VXG

```
vxg = pyarbtools.instruments.VXG(ipAddress port=5025, timeout=10, reset=False)
vxg = pyarbtools.instruments.VXG(ipAddress, apiType='pyvisa', protocol='hislip', port=0,␣
↪timeout=10, reset=False)
```

Arguments

- `ipAddress (str)`: IP address of the instrument.

Keyword Arguments

- `apiType (str)`: API type to use. `'socketscpi'` or `'pyvisa'`, default is `'socketscpi'`.

- `protocol (str)`: Only used when `apiType='pyvisa'`. Chooses which VISA protocol to use. `'hislip'` or `'vxi11'`.
- `port (int)`: Selects socket port when `apiType='socketscpi'`. Selects protocol port when `apiType='pyvisa'`.
- `timeout (int)`: Sets the instrument timeout in seconds. Default is `10`.
- `reset (bool)`: Determines if the instrument will be preset when object is created. `True` or `False`, default is `False`

3.6.1 attributes

These attributes are automatically populated when connecting to the instrument and when calling the `.configure()` method. Generally speaking, they are also the keyword arguments for `.configure()`.

- `instId (str)`: Instrument identifier. Contains instrument model, serial number, and firmware revision.
- `rfState1 | rfState2 (int)`: RF output state per channel. Values are `0` (default) or `1`.
- `modState1 | modState2 (int)`: Modulation state per channel. Values are `0` (default) or `1`.
- `arbState1 | arbState2 (int)`: Internal arb state per channel. Values are `0` (default) or `1`.
- `cf1 | cf2 (float)`: Output carrier frequency in Hz per channel. Values are `10e6` to `44e9`. Default is `1e9`.
- `amp1 | amp2 (float)`: Output power in dBm. Values are `-110` to `+23`. Default is `-100`.
- `alcState1 | alcState2 (int)`: ALC (automatic level control) state per channel. Values are `1` or `0` (default).
- `iqScale1 | iqScale2 (int)`: IQ scale factor in % per channel. Values range from `1` to `100`. Default is `70`.
- `fs1 | fs2 (float)`: Sample rate in Hz per channel. Values `1` to `2.56e9`.
- `refSrc (str)`: Reference clock source. Values are `'int'` (default), or `'ext'`.

```
print(f'VXG Ch 1 Sample Rate: {vxg.fs1} samples/sec.')
>>> VXG Ch 1 Sample Rate: 200000000 samples/sec.
```

3.6.2 configure

```
VXG.configure(**kwargs)
# Example
VXG.configure(rfState1=1, cf1=1e9, amp1=-20)
```

Sets the basic configuration for the VXG and populates class attributes accordingly. It *only* changes the setting(s) for the keyword argument(s) sent by the user.

Arguments

- `rfState1 | rfState2 (int)`: Turns the RF output state on or off per channel. Arguments are `0` (default) or `1`.
- `modState1 | modState2 (int)`: Turns the modulation state on or off per channel. Arguments are `0` (default) or `1`.
- `arbState1 | arbState2 (int)`: Turns the internal arb on or off per channel. Arguments are `0` (default) or `1`.
- `cf1 | cf2 (float)`: Output carrier frequency in Hz per channel. Arguments are `10e6` to `44e9`. Default is `1e9`.

- `amp1 | amp2 (float)`: Output power in dBm per channel. Arguments are `-110` to `+23`. Default is `-100`.
- `alcState1 | alcState2 (int)`: Turns the ALC (automatic level control) on or off per channel. Arguments are `1` or `0` (default).
- `iqScale1 | iqScale2 (int)`: IQ scale factor in % per channel. Argument range is `1` to `100`. Default is `70`.
- `fs1 | fs2 (float)`: Sample rate in Hz per channel. Arguments are `1` to `2.56e9`.
- `refSrc (str)`: Reference clock source. Arguments are `'int'` (default), or `'ext'`.

Returns

- None

3.6.3 download_wfm

```
VXG.download_wfm(wfmData, wfmID='wfm')
```

Defines and downloads a waveform to the default waveform directory on the VXG's hard drive (D:\Users\Instrument\Documents\Keysight\PathWave\SignalGenerator\Waveforms\) and checks that the waveform meets minimum waveform length and granularity requirements. Returns useful waveform identifier.

Arguments

- `wfmData (NumPy array)`: Array of values containing the complex sample pairs in an IQ waveform.
- `wfmID (str)`: Name of the waveform to be downloaded. Default is `'wfm'`.

Returns

- `wfmID` (string): Useful waveform name or identifier. Use this as the waveform identifier for `.play()`.

3.6.4 delete_wfm

```
VXG.delete_wfm(wfmID)
```

Deletes a waveform from the waveform memory.

Arguments

- `wfmID (str)`: Name of the waveform to be deleted.

Returns

- None

3.6.5 clear_all_wfm

```
VXG.clear_all_wfm()
```

Stops playback and deletes all waveforms from the waveform memory.

Arguments

- None

Returns

- None

3.6.6 play

```
VXG.play(wfmID='wfm', ch=1, *args, **kwargs)
```

Selects waveform and activates arb mode, RF output, and modulation.

Arguments

- `wfmID (str)`: Name of the waveform to be loaded. The return value from `.download_wfm()` should be used. Default is `'wfm'`.
- `ch (int)`: Channel out of which the waveform will be played. Default is `1`.

Keyword Arguments

- `rms (float)`: Waveform RMS power calculation. VXG will offset RF power to ensure measured RMS power matches the user-specified RF power. Set to `1.0` for pulses with multiple power levels in a single waveform. This causes the peak power level to match the RF output power setting.

Returns

- None

3.6.7 stop

```
VXG.stop(ch=1)
```

Deactivates arb mode, RF output, and modulation.

Arguments

- `ch (int)`: Channel for which playback will be stopped. Default is `1`.

Returns

- None

3.7 wfmBuilder

In addition to instrument control and communication, PyArbTools allows you to create waveforms and load them into your signal generator or use them as generic signals for DSP work:

```
# Create a sine wave
fs = 12e9
freq = 4e9
wfmFormat = 'real'
real = pyarbtools.wfmBuilder.sine_generator(fs=fs, freq=freq, wfmFormat=wfmFormat)

# Create a digitally modulated signal
fs = 100e6
modType = 'qam64'
symRate = 20e6
iq = pyarbtools.wfmBuilder.digmod_generator(fs=fs, modType=modType, symRate=symRate)

# Export waveform to csv file
```

(continues on next page)

(continued from previous page)

```
fileName = 'C:\\temp\\waveforms\\20MHz_64QAM.csv'
pyarbtools.wfmBuilder.export_wfm(iq, fileName)
```

3.7.1 export_wfm

```
export_wfm(data, fileName, vsaCompatible=False, fs=0)
```

Takes in waveform data and exports it to a csv file as plain text.

Arguments

- `data (NumPy array)`: Waveform data to be exported.
- `fileName (str)`: Full absolute file name where the waveform will be saved. (should end in `".csv"`)
- `vsaCompatible (bool)`: Determines VSA compatibility. If `True`, adds the `XDelta` field to the beginning of the file and allows VSA to recall it as a recording.
- `fs (float)`: Sample rate originally used to create the waveform. Default is `0`, so this should be entered manually.

Returns

- None

3.7.2 import_mat

```
import_mat(fileName, targetVariable='data')
```

Imports waveform data from .mat file. Detects array data type, and accepts data arrays in 1D real or complex, or 2 separate 1D arrays for I and Q.

Arguments

- `fileName (str)`: Full absolute file name for .mat file.
- `targetVariable (str)`: User-specifiable name of variable in .mat file containing waveform data.

Returns

- **`(dict):`**
  - `data` (NumPy ndarray): Array of waveform samples.
  - `fs` (float): Sample rate of imported waveform.
  - `wfmID (str)`: Waveform name.
  - `wfmFormat (str)`: Waveform format (`iq` or `real`).

3.7.3 zero_generator

```
zero_generator(fs=100e6, numSamples=1024, wfmFormat='iq')
```

Generates a waveform filled with the value `0`.

Arguments

- `fs (float)`: Sample rate used to create the signal in Hz. Argument is a float. Default is `50e6`.
- `numSamples (int)`: Length of the waveform in samples.
- `wfmFormat (str)`: Waveform format. Arguments are `'iq'` (default) or `'real'`.

Returns

- `(NumPy array)`: Array containing the complex or real values of the zero waveform.

3.7.4 sine_generator

```
sine_generator(fs=100e6, freq=0, phase=0, wfmFormat='iq', zeroLast=False)
```

Generates a sine wave with configurable frequency and initial phase at baseband or RF.

Arguments

- `fs (float)`: Sample rate used to create the signal in Hz. Argument is a float. Default is `50e6`.
- `freq (float)`: Sine wave frequency.
- `phase (float)`: Initial phase offset. Argument range is `0` to `360`.
- `wfmFormat (str)`: Waveform format. Arguments are `'iq'` (default) or `'real'`.
- `zeroLast (bool)`: Allows user to force the last sample point to `0`. Default is `False`.

Returns

- `(NumPy array)`: Array containing the complex or real values of the sine wave.

3.7.5 am_generator

```
am_generator(fs=100e6, amDepth=50, modRate=100e3, cf=1e9, wfmFormat='iq', zeroLast=False)
```

Generates a linear sinusoidal AM signal of specified depth and modulation rate at baseband or RF.

Arguments

- `fs (float)`: Sample rate used to create the signal in Hz. Default is `50e6`.
- `amDepth (int)`: Depth of AM in %. Argument range is `0` to `100`. Default is `50`.
- `modRate (float)`: AM rate in Hz. Argument range is `0` to `fs/2`. Default is `100e3`.
- `cf (float)`: Center frequency for `'real'` format waveforms. Default is `1e9`.
- `wfmFormat (str)`: Waveform format. Arguments are `'iq'` (default) or `'real'`.
- `zeroLast (bool)`: Allows user to force the last sample point to `0`. Default is `False`.

Returns

- `(NumPy array)`: Array containing the complex or real values of the AM waveform.

3.7.6 cw_pulse_generator

```
wfmBuilder.cw_pulse_generator(fs=100e6, pWidth=10e-6, pri=100e-6, freqOffset=0, cf=1e9,␣
↪wfmFormat='iq', zeroLast=False, ampScale=100)
```

Generates an unmodulated CW (continuous wave) pulse at baseband or RF.

Arguments

- `fs (float)`: Sample rate used to create the signal in Hz. Default is `100e6`.
- `pWidth (float)`: Length of the pulse in seconds. Default is `10e-6`. The pulse width will never be shorter than `pWidth`, even if `pri < pWidth`.
- `pri (float)`: Pulse repetition interval in seconds. Default is `100e-6`. If `pri > pWidth`, the dead time will be included in the waveform.
- `freqOffset (float)`: Frequency offset from carrier frequency in Hz. Default is `0`.
- `cf (float)`: Center frequency for `'real'` format waveforms. Default is `1e9`.
- `wfmFormat (str)`: Waveform format. Arguments are `'iq'` (default) or `'real'`.
- `zeroLast (bool)`: Allows user to force the last sample point to `0`. Default is `False`.
- `ampScale (int)`: Sets the linear voltage scaling of the waveform samples. Default is `100`. Range is `0` to `100`.

Returns

- `iq/real (NumPy array)`: Array containing the complex or real values of the CW pulse.

3.7.7 chirp_generator

```
wfmBuilder.chirp_generator(fs=100e6, pWidth=10e-6, pri=100e-6, chirpBw=20e6, cf=1e9,␣
↪wfmFormat='iq', zeroLast=False)
```

Generates a symmetrical linear chirped pulse at baseband or RF. Chirp direction is determined by the sign of chirpBw (pos=up chirp, neg=down chirp).

Arguments

- `fs (float)`: Sample rate used to create the signal in Hz. Default is `100e6`.
- `pWidth (float)`: Length of the pulse in seconds. Default is `10e-6`. The pulse width will never be shorter than `pWidth`, even if `pri < pWidth`.
- `pri (float)`: Pulse repetition interval in seconds. Default is `100e-6`. If `pri > pWidth`, the dead time will be included in the waveform.
- `chirpBw (float)`: Total bandwidth of the chirp. Frequency range of resulting signal is `-chirpBw/2` to `chirpBw/2`. Default is `20e6`.
- `cf (float)`: Center frequency for `'real'` format waveforms. Default is `1e9`.
- `wfmFormat (str)`: Waveform format. Arguments are `'iq'` (default) or `'real'`.
- `zeroLast (bool)`: Allows user to force the last sample point to `0`. Default is `False`.

Returns

- `iq/real (NumPy array)`: Array containing the complex or real values of the chirped pulse.

3.7.8 barker_generator

```
wfmBuilder.barker_generator(fs=100e6, pWidth=100e-6, pri=100e-6, code='b2', cf=1e9,␣
↪wfmFormat='iq', zeroLast=False)
```

Generates a Barker phase coded pulsed signal at RF or baseband. See Wikipedia article for more information on Barker coding.

Arguments

- `fs (float)`: Sample rate used to create the signal in Hz. Default is `100e6`.
- `pWidth (float)`: Length of the pulse in seconds. Default is `10e-6`. The pulse width will never be shorter than `pWidth`, even if `pri < pWidth`.
- `pri (float)`: Pulse repetition interval in seconds. Default is `100e-6`. If `pri > pWidth`, the dead time will be included in the waveform.
- `code (str)`: Barker code order. Arguments are `'b2'` (default), `'b3'`, `'b41'`, `'b42'`, `'b5'`, `'b7'`, `'b11'`, or `'b13'`.
- `cf (float)`: Center frequency for `'real'` format waveforms. Default is `1e9`.
- `wfmFormat (str)`: Waveform format. Arguments are `'iq'` (default) or `'real'`.
- `zeroLast (bool)`: Allows user to force the last sample point to `0`. Default is `False`.

Returns

- `iq/real (NumPy array)`: Array containing the complex or real values of the barker pulse.

3.7.9 multitone_generator

```
multitone_generator(fs=100e6, spacing=1e6, num=11, phase='random', cf=1e9, wfmFormat='iq
↪')
```

Generates a multitone_generator signal with given tone spacing, number of tones, sample rate, and phase relationship.

Arguments

- `fs (float)`: Sample rate used to create the signal in Hz. Default is `100e6`.
- `spacing (float)`: Tone spacing in Hz. There is currently no limit to `spacing`, so beware of the compilation time for small spacings and beware of aliasing for large spacings.
- `num (int)`: Number of tones. There is currently no limit to `num`, so beware of long compilation times for large number of tones.
- `phase (str)`: Phase relationship between tones. Arguments are `'random'` (default), `'zero'`, `'increasing'`, or `'parabolic'`.
- `cf (float)`: Center frequency for `'real'` format waveforms. Default is `1e9`.
- `wfmFormat (str)`: Waveform format. Arguments are `'iq'` (default) or `'real'`.

Returns

- `iq/real (NumPy array)`: Array containing the complex or real values of the multitone_generator signal.

3.7.10 digmod_generator

```
def digmod_generator(fs=10, symRate=1, modType='bpsk', numSymbols=1000, filt=
↪'raisedcosine', alpha=0.35, wfmFormat='iq', zeroLast=False, plot=False)
```

Generates a baseband modulated signal with a given modulation type and transmit filter using random data.

Arguments

- `fs (float)`: Sample rate used to create the waveform in samples/sec.
- `symRate (float)`: Symbol rate in symbols/sec.
- `modType (str)`: Type of modulation. ('bpsk', 'qpsk', 'psk8', 'psk16', 'apsk16', 'apsk32', 'apsk64', 'qam16', 'qam32', 'qam64', 'qam128', 'qam256')
- `numSymbols (int)`: Number of symbols to put in the waveform.
- `filt (str)`: Pulse shaping filter type. ('raisedcosine' or 'rootraisedcosine')
- `alpha (float)`: Pulse shaping filter excess bandwidth specification. Also known as roll-off factor, alpha, or beta. (`0 - 1.0`)
- `wfmFormat (str)`: Determines type of waveform. Currently only 'iq' format is supported.
- `zeroLast (bool)`: Enable or disable forcing the last sample point to 0.
- `plot (bool)`: Enable or disable plotting of final waveform in time domain and constellation domain.

NOTE - The ring ratios for APSK modulations are as follows:

- 16-APSK: R1 = 1, R2 = 2.53
- 32-APSK: R1 = 1, R2 = 2.53, R3 = 4.3
- 64-APSK: R1 = 1, R2 = 2.73, R3 = 4.52, R4 = 6.31

Returns

- `(NumPy array)`: Array containing the complex values of the digitally modulated signal.

3.7.11 iq_correction

```
iq_correction(iq, inst, vsaIPAddress='127.0.0.1', vsaHardware='"Analyzer1"', cf=1e9,␣
↪osFactor=4, thresh=0.4, convergence=2e-8):
```

Creates a 16-QAM signal from a signal generator at a user-selected center frequency and sample rate. Symbol rate and effective bandwidth of the calibration signal is determined by the oversampling rate in VSA. Creates a VSA instrument, which receives the 16-QAM signal and extracts & inverts an equalization filter and applies it to the user-defined waveform.

Arguments

- `iq (NumPy array)`: Array contianing the complex values of the signal to be corrected.
- `inst (pyarbtools.instrument.XXX)`: Instrument class of the generator to be used in the calibration. Must already be connected and configured. `inst.fs` is used as the basis for the calibration and `inst.play()` method is used.
- `vsaIPAddress (str)`: IP address of the VSA instance to be used in calibration. Default is `'127.0.0.1'`.
- `vsaHardware (str)`: Name of the hardware to be used by VSA. Name must be surrounded by double quotes ("). Default is `'"Analyzer1"'`.

- `cf (float)`: Center frequency at which calibration takes place. Default is `1e9`.
- `osFactor (int)`: Oversampling factor used by the digital demodulator in VSA. The larger the value, the narrower the bandwidth of the calibration. Effective bandwidth is roughly `inst.fs / osFactor * 1.35`. Arguments are `2`, `4` (default), `5`, `10`, or `20`.
- `thresh (float)`: Defines the target EVM value that should be reached before extracting equalizer impulse response. Argument range is `0` to `1.0`. Default is `0.4`. Low values take longer to settle but result in better calibration.
- `convergence (float)`: Equalizer convergence value. Argument should be << 1. Default is `2e-8`. High values settle more quickly but may become unstable. Lower values take longer to settle but tend to have better stability.

Returns

- `(NumPy array)`: Array containing the complex values of corrected signal.

3.8 vsaControl

To use/control an instance of Keysight 89600 VSA software, create an instance of `pyarbtools.vsaControl.VSA` and enter VSA's IP address as the first argument. There are additional keyword arguments you can add to set things like `port`, `timeout`, and `reset`:

```
# Example
vsa = pyarbtools.vsaControl.VSA('127.0.0.1')
```

Just like all the `pyarbtools.instruments` classes, the VSA class is built on a robust socket connection that allows the user to send SCPI commands/queries, send/receive data using IEEE 488.2 binary block format, check for errors, and gracefully disconnect from the instrument. Methods were named so that those coming from using a VISA interface would be familiar with syntax. This architectural decision to include an open SCPI interface was made to provide additional flexibility for users who need to use specific setup commands *not* covered by built-in functions:

```
# Example
vsa.write('*RST')
instID = vsa.query('*IDN?')
vsa.acquire_single()
traceData = vsa.binblockread('trace1:data:y?')
vsa.disconnect()
```

When an instance of `VSA` is created, PyArbTools connects to the software at the IP address given by the user and sends a few queries. The `VSA` class has a `reset` keyword argument that causes the software to perform a default setup prior to running the rest of the code. It's set to `False` by default to prevent unwanted settings changes.

`VSA` currently supports two measurement types: `vector` and `ddemod` (digital demodulation) and includes a configuration method for each measurement. They provide keyword arguments to configure selected settings for the measurements *and set relevant class attributes* so that the user knows how the analysis software is configured and can use those variables in code without having to send a SCPI query to determine values:

```
vsa.configure_ddemod(modType='bpsk', symRate=10e6, measLength=128)
print(f'Modulation type is {vsa.modType}.')
print(f'Symbol rate is {vsa.symRate} symbols/sec.')
```

3.9 VSA

```
pyarbtools.vsaControl.VSA(ipAddress port=5025, timeout=10, reset=False, vsaHardware=None)
```

3.9.1 attributes

These attributes are automatically populated when connecting to the instrument and when calling the `.configure_ddemod()` and `.configure_vector()` methods. Generally speaking, they are also the keyword arguments for the `.configure_***()` methods.

- `instId (str)`: Instrument identifier. Contains instrument model, serial number, and firmware revision.
- `cf (float)`: Analyzer center frequency in Hz.
- `amp (float)`: Reference level/vertical range in dBm.
- `span (float)`: Analyzer span in Hz.
- `hw (str)`: Identifier string for acquisition hardware used by VSA.
- `meas (str)`: Measurement type ('vector', 'ddemod' currently supported).
- `modType (str)`: String defining digital modulation format.
- `symRate (float)`: Symbol rate in symbols/sec.
- `measFilter (str)`: Sets the measurement filter type.
- `refFilter (str)`: Sets the reference filter type.
- `filterAlpha (float)`: Filter alpha/rolloff factor. Must be between 0 and 1.
- `measLength (int)`: Measurement length in symbols.
- `eqState (bool)`: Turns the equalizer on or off.
- `eqLength (int)`: Length of the equalizer filter in symbols.
- `eqConvergence (float)`: Equalizer convergence factor.
- `rbw (float)`: Resolution bandwidth in Hz.
- `time (float)`: Analysis time in sec.

3.9.2 acquire_continuous

```
VSA.acquire_continuous()
```

Begins continuous acquisition in VSA using SCPI commands.

Arguments

- None

Returns

- None

3.9.3 acquire_single

```
VSA.acquire_single()
```

Sets single acquisition mode and takes a single acquisition in VSA using SCPI commands.

Arguments

- None

Returns

- None

3.9.4 stop

```
VSA.stop()
```

Stops acquisition in VSA using SCPI commands.

Arguments

- None

Returns

- None

3.9.5 autorange

```
VSA.autorange()
```

Executes an amplitude autorange in VSA and waits for it to complete using SCPI commands.

Arguments

- None

Returns

- None

3.9.6 set_hw

```
VSA.set_hw(hw)
```

Sets and reads hardware configuration for VSA. Checks to see if selected hardware is valid.

Arguments

- `hw (str)`: Identifier string for acquisition hardware used for VSA

Returns

- None

3.9.7 set_cf

```
VSA.set_cf(cf)
```

Sets and reads center frequency for VSA using SCPI commands.

Arguments

- `cf (float)`: Analyzer center frequency in Hz.

Returns

- None

3.9.8 set_amp

```
VSA.set_amp(amp)
```

Sets and reads reference level/vertical range for VSA using SCPI commands.

Arguments

- `amp (float)`: Analyzer reference level/vertical range in dBm.

Returns

- None

3.9.9 set_span

```
VSA.set_span(span)
```

Sets and reads span for VSA using SCPI commands.

Arguments

- `span (float)`: Analyzer span in Hz.

Returns

- None

3.9.10 set_measurement

```
VSA.set_amp(meas)
```

Sets and reads measurement type in VSA using SCPI commands.

Arguments

- `meas (srt)`: Selects measurement type ('vector', 'ddemod' currently supported).

Returns

- None

3.9.11 configure_ddemod

```
VSA.configure_ddemod(**kwargs)
# Example
VSA.configure_ddemod(cf=1e9, modType='qam16', symRate=1e6)
```

Configures digital demodulation settings in VSA using SCPI commands.

Keyword Arguments

- `cf (float)`: Analyzer center frequency in Hz.
- `amp (float)`: Analyzer reference level/vertical range in dBm.
- `span (float)`: Analyzer span in Hz.
- `modType (str)`: String defining digital modulation format.
- `symRate (float)`: Symbol rate in symbols/sec.
- `measFilter (str)`: Sets the measurement filter type.
- `refFilter (str)`: Sets the reference filter type.
- `filterAlpha (float)`: Filter alpha/rolloff factor. Must be between 0 and 1.
- `measLength (int)`: Measurement length in symbols.
- `eqState (bool)`: Turns the equalizer on or off.
- `eqLength (int)`: Length of the equalizer filter in symbols.
- `eqConvergence (float)`: Equalizer convergence factor.

Returns

- None

3.9.12 configure_vector

```
VSA.configure_vector(**kwargs)
# Example
VSA.configure_vector(cf=1e9, span=40e6, rbw=100e3)
```

Configures vector measurement mode in VSA using SCPI commands. Note that the `time` and `rbw` settings are interconnected. If you set both, the latter setting will override the first one set.

Keyword Arguments

- `cf (float)`: Analyzer center frequency in Hz.
- `amp (float)`: Analyzer reference level/vertical range in dBm.
- `span (float)`: Analyzer span in Hz.
- `rbw (float)`: Resolution bandwidth in Hz.
- `time (float)`: Analysis time in sec.

Returns

- None

3.9.13 recall_recording

```
VSA.recall_recording(fileName, fileFormat='csv')
```

Recalls a data file as a recording in VSA using SCPI commands.

Arguments

- `fileName (str)`: Full absolute file name of the recording to be loaded.
- `fileFormat (str)`: Format of recording file. ('CSV', 'E3238S', 'MAT', 'MAT7', 'N5110A', 'N5106A', 'SDF', 'TEXT')

Returns

- None

3.9.14 sanity_check

```
VSA.sanity_check()
```

Prints out measurement-context-sensitive user-accessible class attributes

Arguments

- None

Returns

- None

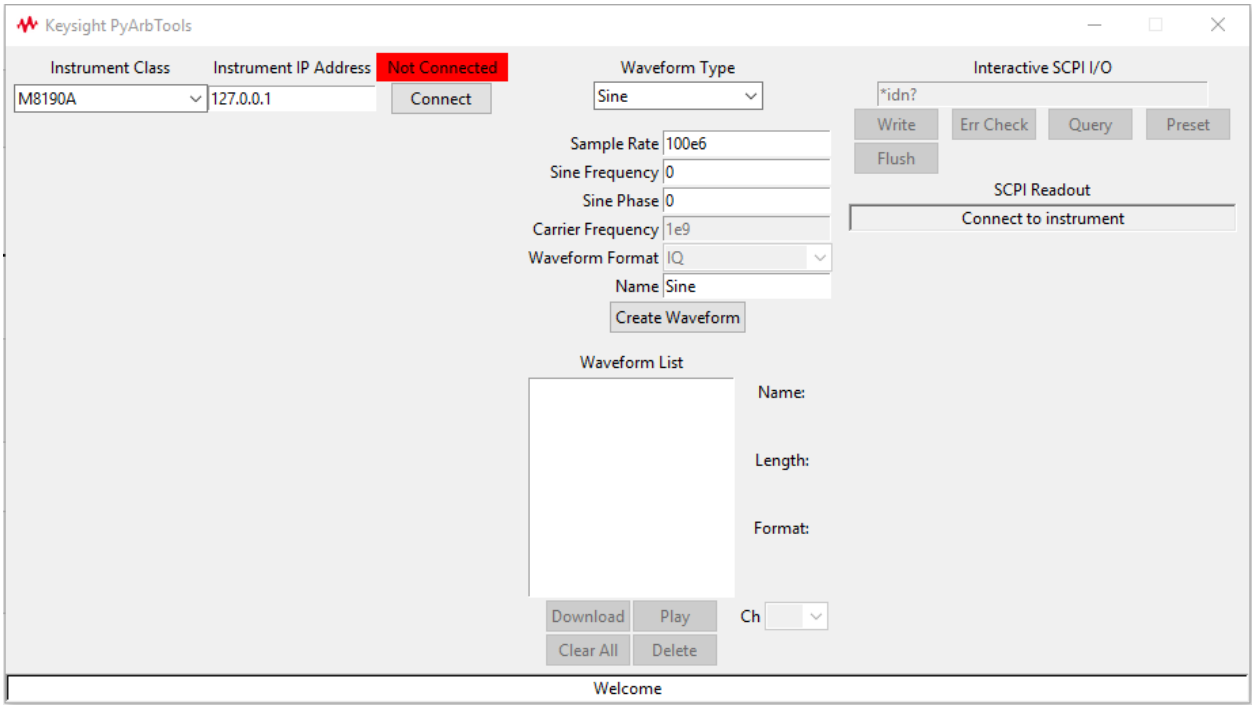
3.9.15 GUI

```
pyarbtools.gui.main()
```

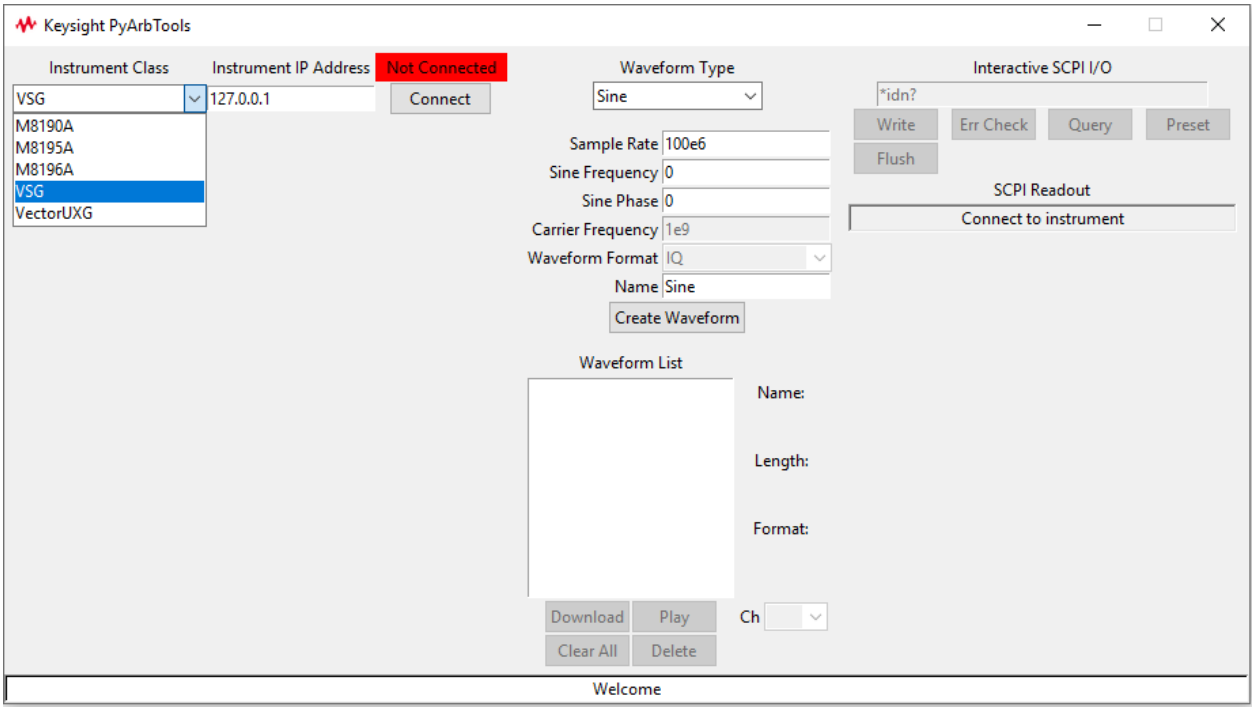
The PyArbTools GUI is experimental. Please provide feedback and feature requests.

Quick Guide

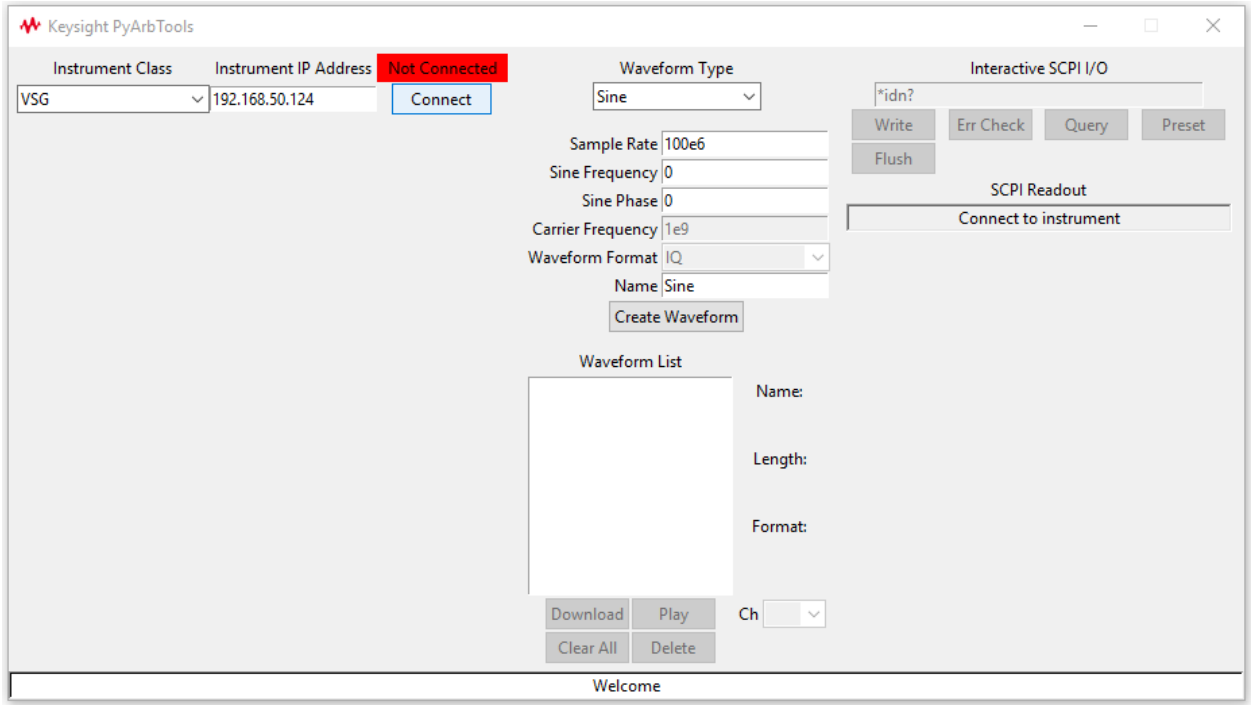
This is what you will see upon starting the GUI.



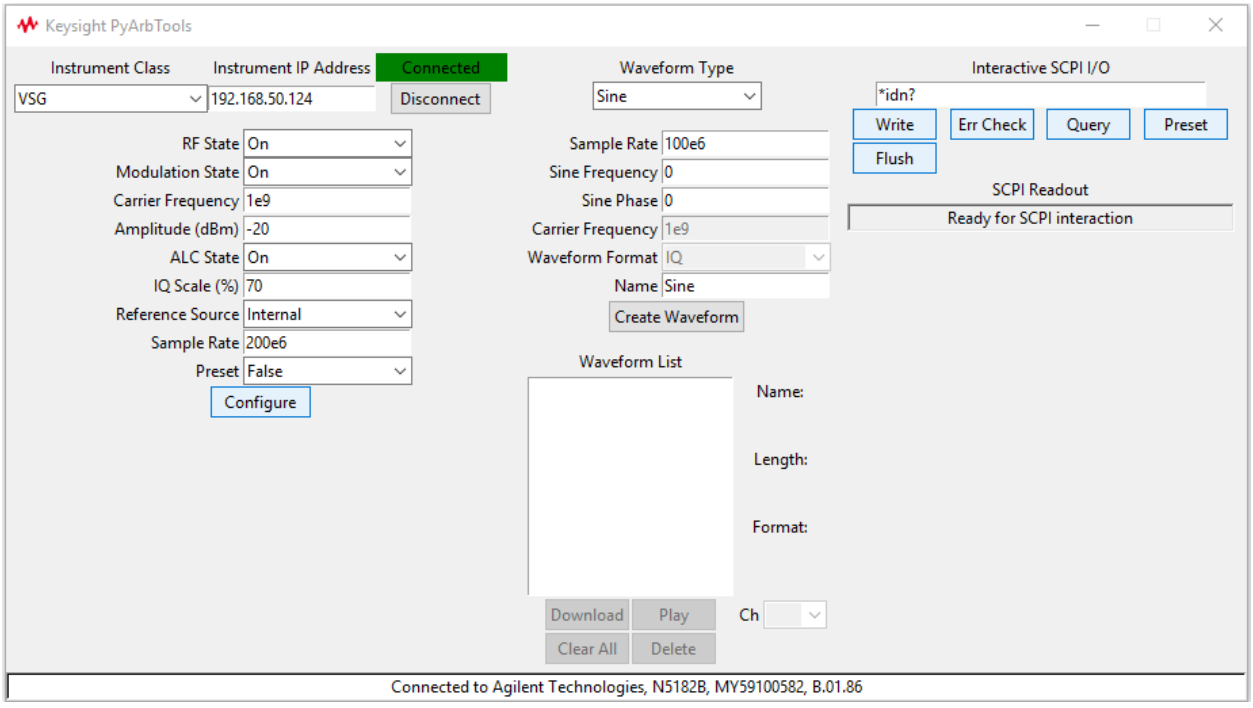
Select an **Instrument Class** from the dropdown menu. For a list of supported equipment, go to the top of this page.



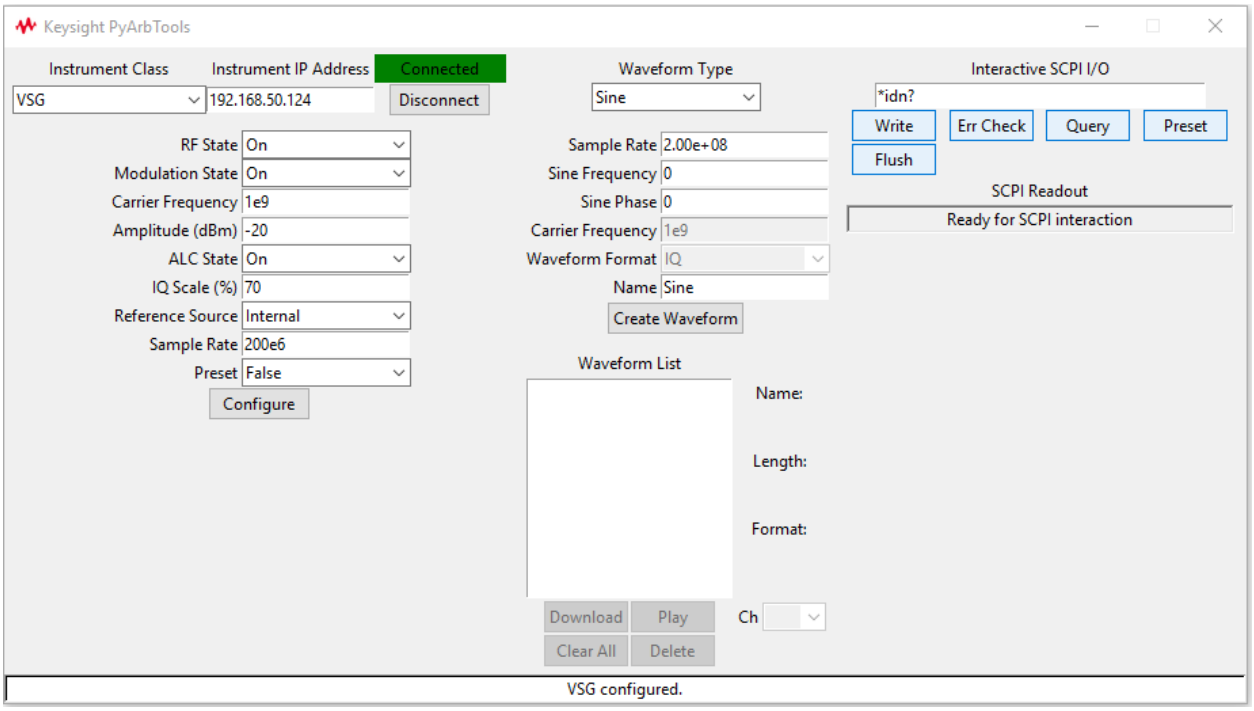
Enter the IP address of your instrument and click **Connect**.



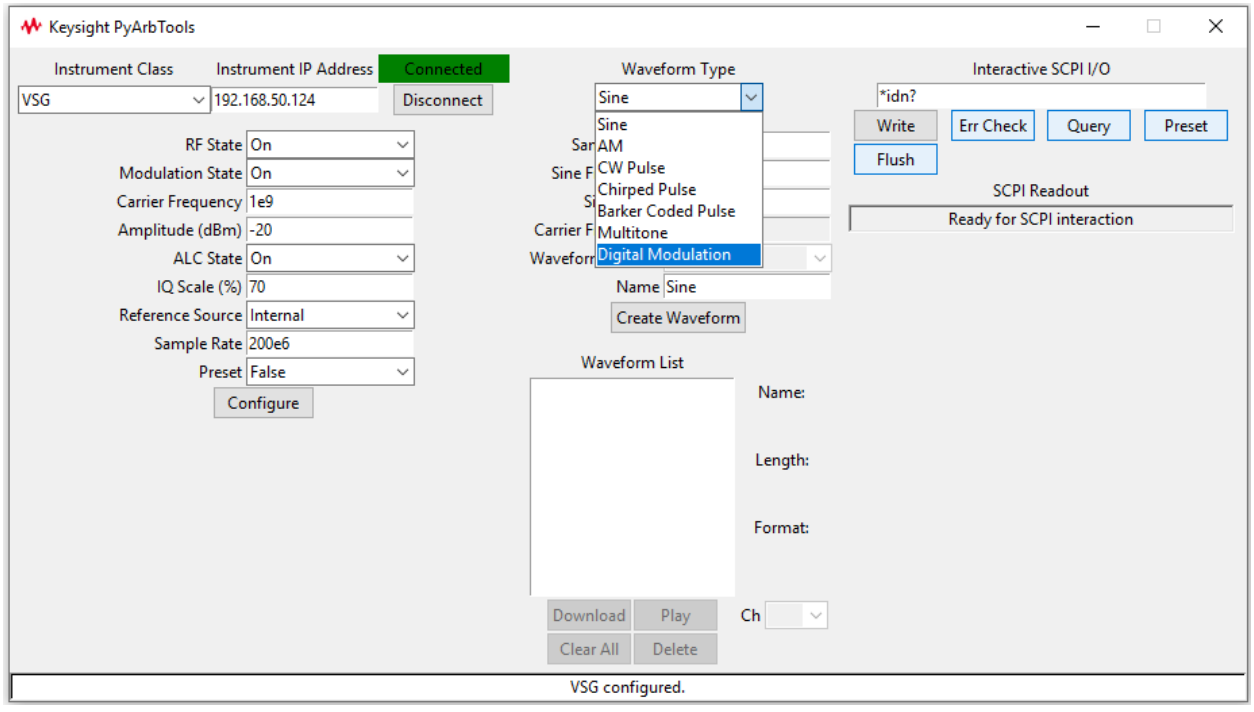
Choose the relevant hardware settings in your instrument and click **Configure**.



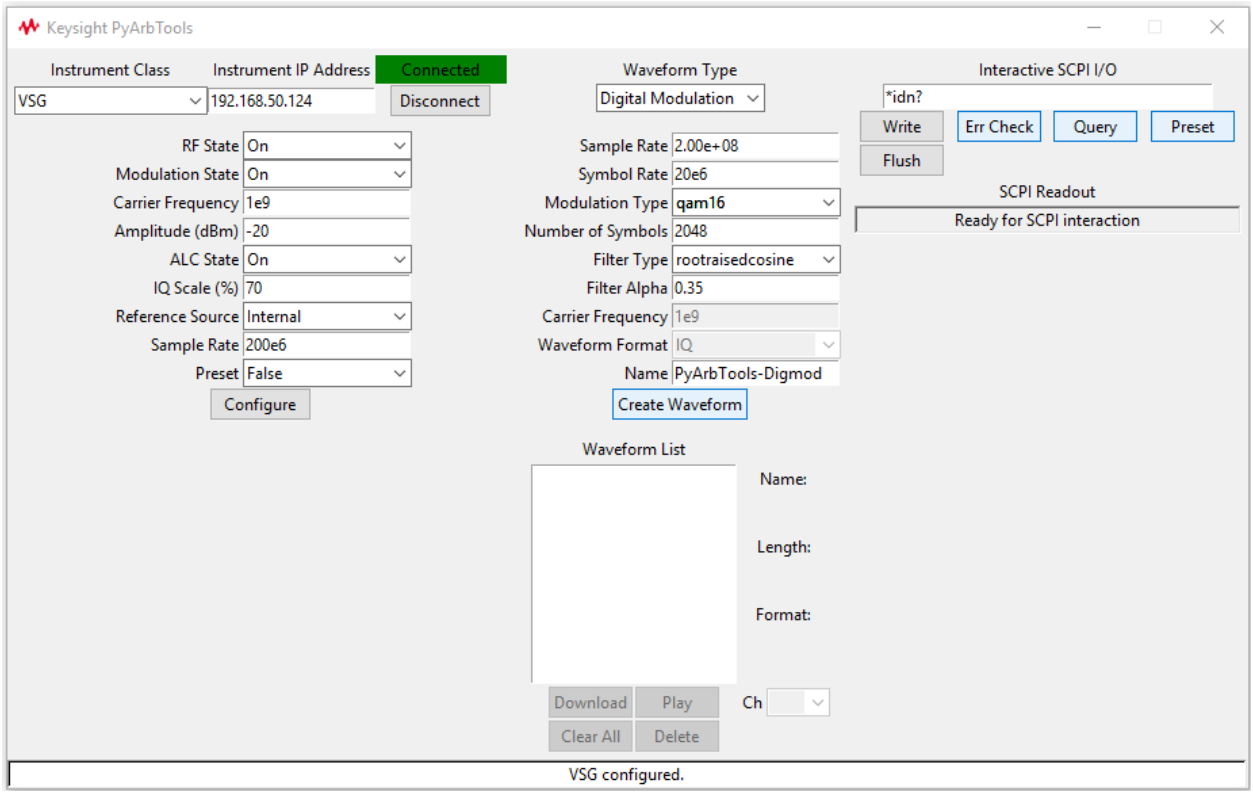
You'll see the status bar along the bottom shows a message on config status.



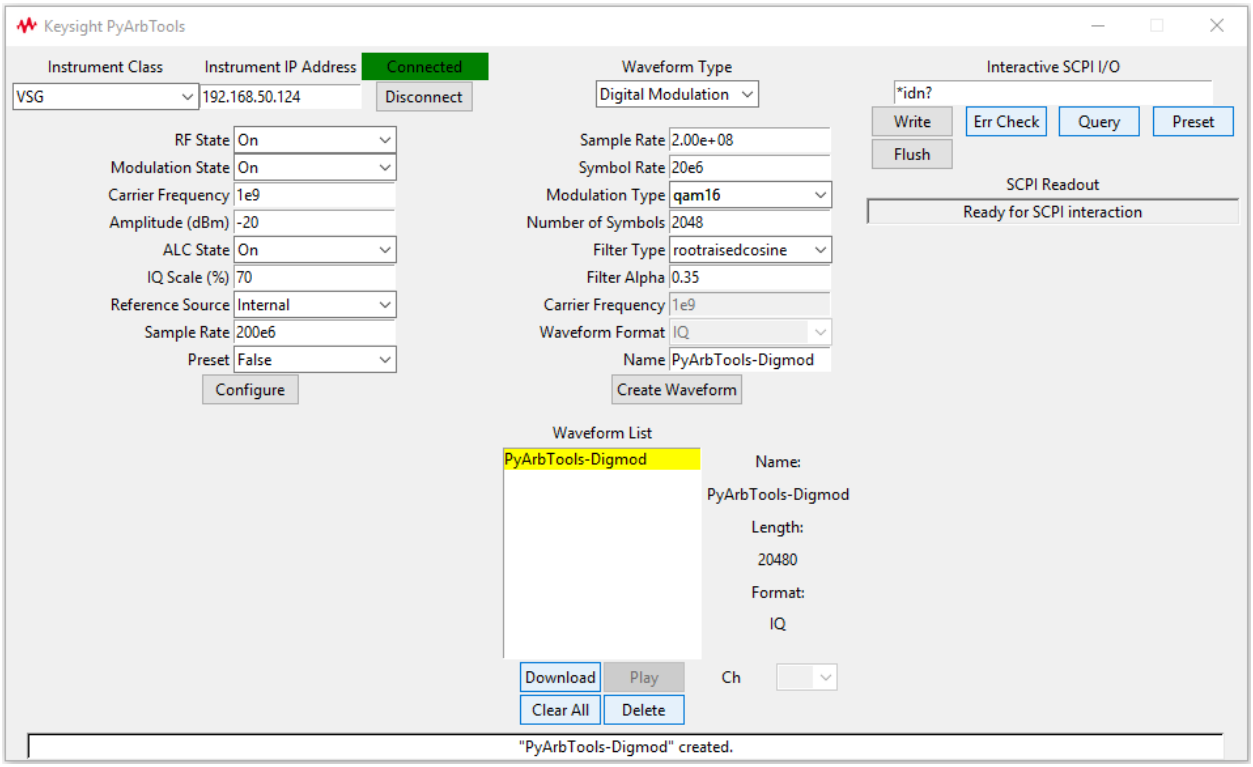
Now we can start creating waveforms. Pick a **Waveform Type** from the dropdown menu.



Choose the specific settings for your waveform and click **Create Waveform**.

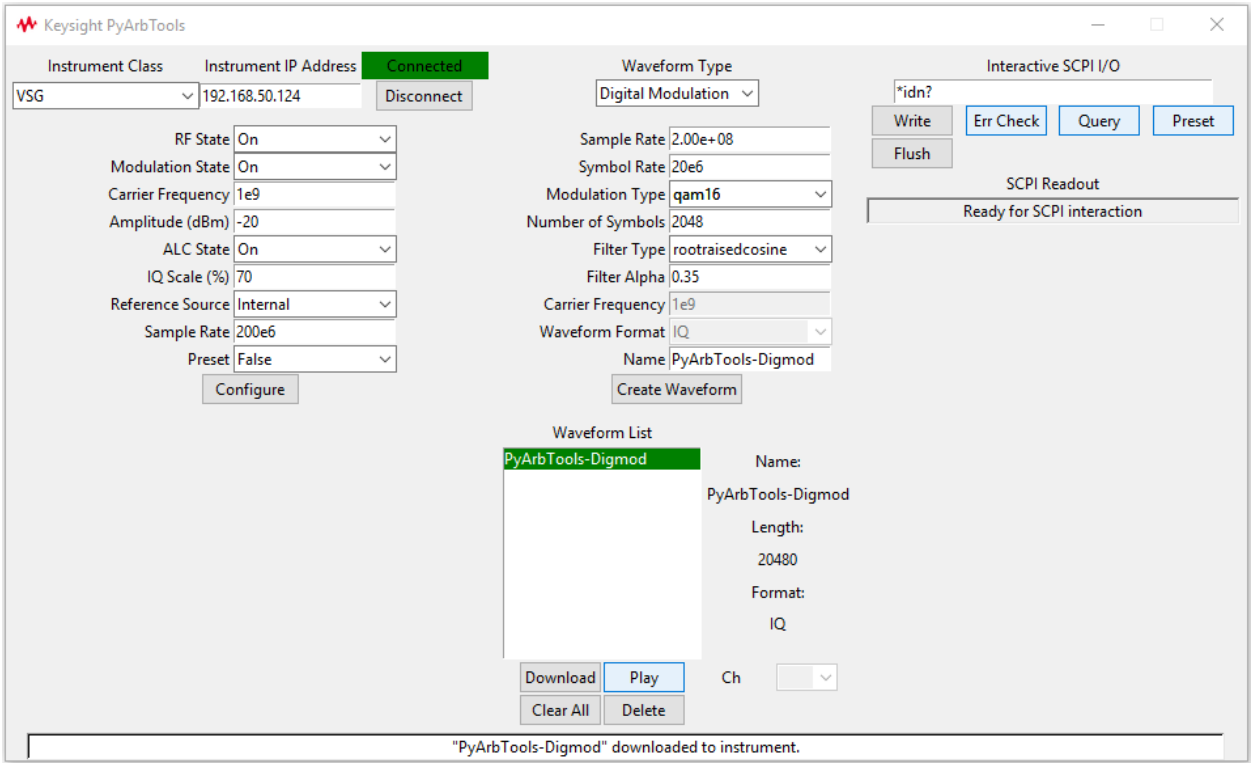


You'll now see an entry in with a yellow background in the **Waveform List**. This means it's been created but not downloaded to the signal generator.

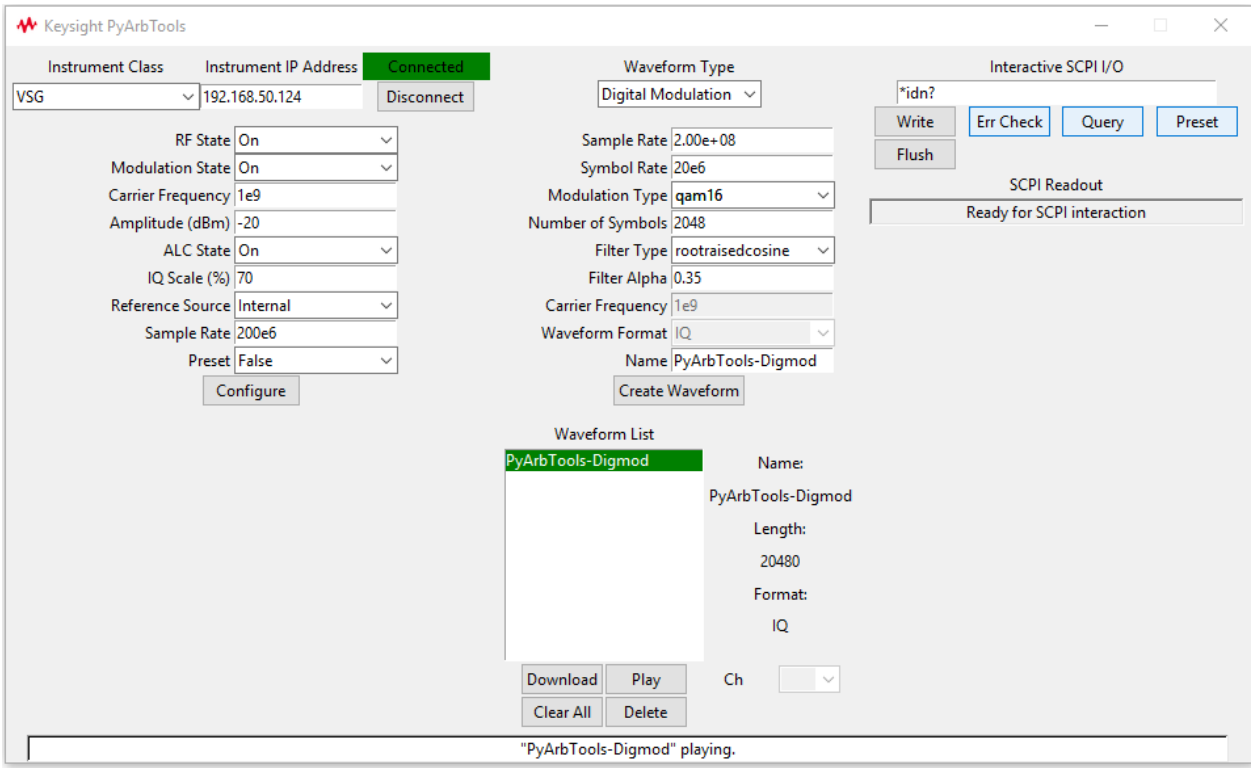


Click **Download** and the yellow entry will turn to green. This means the waveform has been downloaded to the signal

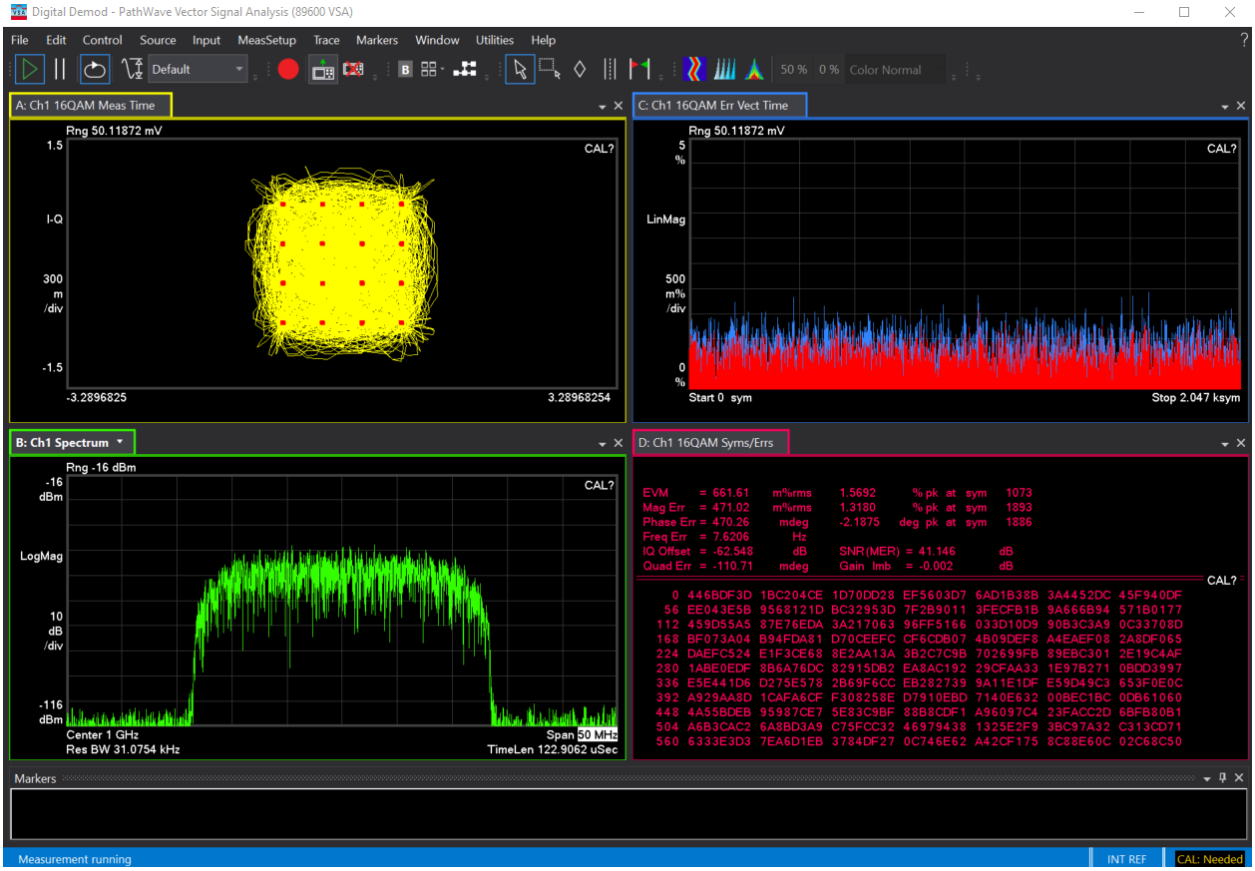
generator.



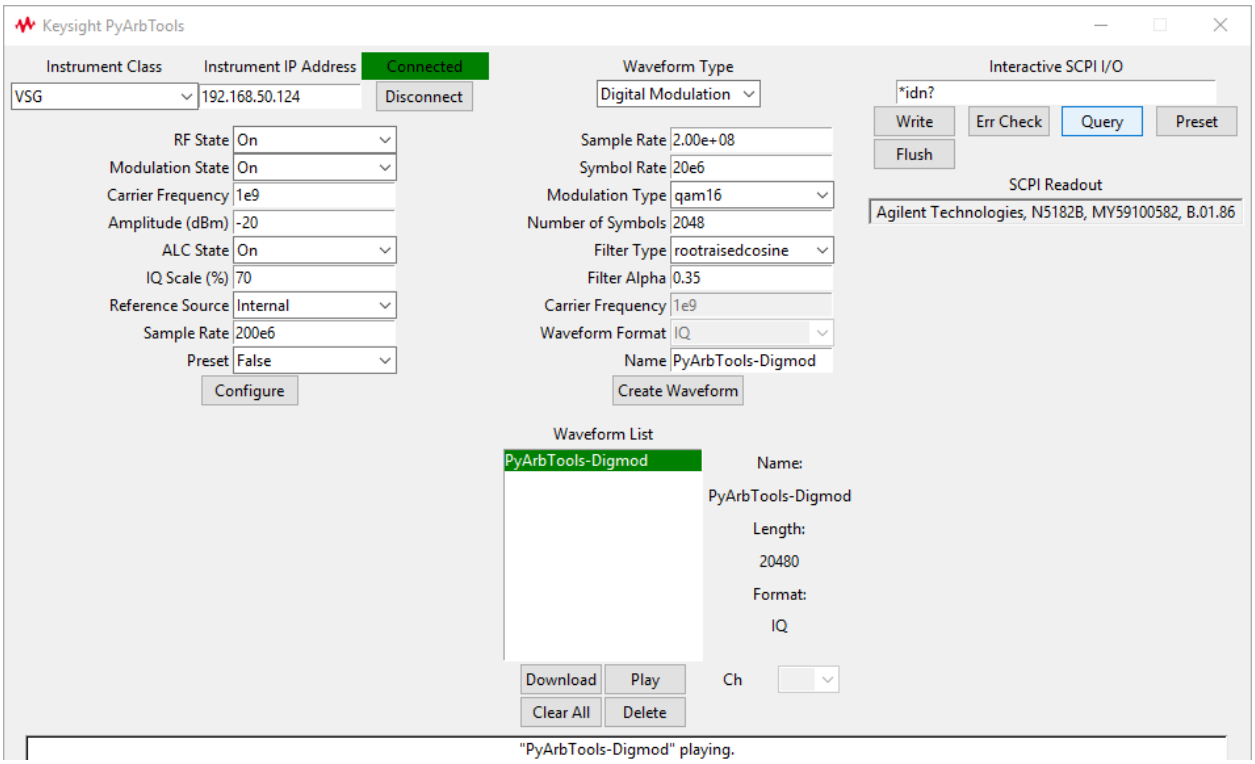
Click **Play** to start playback out of the signal generator.



Below are the results of the steps we just took in Keysight's VSA software.



You can also use PyArbTools as an **Interactive SCPI I/O** tool. Below are the results of the '*IDN?' query.



CHAPTER

FOUR

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/morgan-at-keysight/pyarbtools/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

pyarbtools could always use more documentation, whether as part of the official pyarbtools docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/morgan-at-keysight/pyarbtools/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *pyarbtools* for local development.

1. Fork the *pyarbtools* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pyarbtools.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyarbtools
$ cd pyarbtools/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8:

```
$ flake8 pyarbtools
```

   To get flake8, just pip install it into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

4.4 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

CHAPTER

FIVE

CREDITS

5.1 Development Lead

- Morgan Allison <morgan.allison@keysight.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER

SIX

HISTORY

6.1 0.0.7 (2018-10-09)

- First release on PyPI.

6.2 0.0.8 (2018-11-26)

- First major update. Added `wfmBuilder.iq_correction()`, which utilizes Keysight's VSA software and a Keysight receiver (either an oscilloscope or signal analyzer) and applies a digital predistortion filter to a waveform to flatten amplitude and phase response.

6.3 0.0.10 (2018-12-10)

- Added `multitone_generator` method. Added 128-QAM and 256-QAM. Updated index to include a better intro to the project.

6.4 0.0.11 (2019-01-23)

- Fixed bugs with `multitone_generator` method. Added AM modulator.

6.5 0.0.12 (2019-02-20)

- Expanded `VSG` class to include M9381A and M9383A without changing public API.

6.6 0.0.13 (2019-04-19)

- Added GUI. Adjusted several functions to streamline waveform creation and download between all instrument types.

6.7 0.0.14 (2019-11-12)

- Added `pri` argument to pulse creation methods.

6.8 2020.04.0 (2020-04-29)

- Removed communications.py and replaced it with external module `socketscpi`. `.configure()` methods now use `**kwargs` to prevent the function from changing any settings not explicitly specified back to default values. Changed multitone_generator waveform creation method to frequency domain. Significant updates to documentation and in-code comments. Changed from semantic versioning to calendar versioning.

6.9 2020.05.0 (2020-05-13)

- Changed name of `digmod_prbs_generator()` to `digmod_generator()`. Overhauled digitally modulated waveform creation function, fixing bugs and producing better signal fidelity. Added `vsaControl.py`, which allows the user to control an instance of Keysight 89600 VSA software for waveform analysis.

6.10 2020.06.0 (2020-06-01)

- Added 16-, 32-, and 64-APSK modulation types to wfmBuilder.py. Moved PDW-building function definitions from instruments.py to pdwBuilder.py. Updated documentation.

6.11 2020.07.1 (2020-07-06)

- BUGFIX: corrected a problem with the 16-QAM modulator in `digmod_generator()`. Fixed a math error in power level calculations in PDW functions.

6.12 2020.07.3 (2020-07-31)

- Dramatically improved resampling and wraparound handling in `digmod_generator()`.

6.13 2020.08.1 (2020-08-03)

- BROKEN: Updated GUI to reflect changes to instrument configuration functions and improvements to wfmBuilder functions. You can now access `pyarbtools.pdwBuilder` directly. Improved version updating.

6.14 2020.08.2 (2020-08-06)

- Fixed issue where the GUI icon was not included with the PyArbTools package, which prevented the GUI from running.

6.15 2020.09.1 (2020-09-03)

- Added `VXG` instrument class for controlling the M9384B VXG signal generators.

6.16 2020.09.2 (2020-09-08)

- Added `import_mat()` method to wfmBuilder.py that imports .mat files containing waveform data and optional metadata.

6.17 2021.02.1 (2021-02-22)

- Added `ampScale` as an argument to `cw_pulse_generator()`.

6.18 2021.02.2 (2021-02-23)

- Added an `rms` keyword argument to `VXG.play()` that allows the user to manually override the RMS power calculation made by the VXG. This is an advanced feature primarily developed for waveforms containing pulsed signals with different power levels.

6.19 2021.02.3 (2021-02-24)

- Added long-awaited sequencer functionality to `M8190A`. Added `create_sequence()`, `insert_wfm_in_sequence()`, and `insert_idle_in_sequence()` methods to `M8190A`. Added `zero_generator()` function to `wfmBuilder`.

6.20 2021.02.4 (2021-02-26)

- Fixed bug with `ampScale` argument in `cw_pulse_generator()`.

6.21 2021.06.2 (2021-06-26)

- Fixed marker generation functionality in M8190A.

6.22 2021.11.1 (2021-11-01)

- Fixed broken links in ReadMe
- Fixed timeout issue with single-channel VXGs
- Relaxed `VXG.configure()` type checking

6.23 2022.02.1 (2022-02-01)

- Added `get_iq_data()` method to `VSA` object
- Added two new functions in `examples.py`

6.24 2022.03.2 (2022-03-23)

- Added `VMA` class to `vsaControl`. Currently only supports custom OFDM demod using .xml import.

6.25 2022.04.1 (2022-04-19)

- Cleaned up documentation for `vsaControl`.

6.26 2022.11.1 (2022-11-10)

- Removed all features for the UXG.

6.27 2022.11.2 (2022-11-10)

- Fixed issues with documentation.

6.28 2023.06.1 (2023-06-13)

- Implemented `SignalGeneratorBase` class to support either `socketscpi` or `pyvisa` for instrument communications.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search