<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Packages</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Devices</td>
<td>49</td>
</tr>
<tr>
<td>8.2</td>
<td>GUI</td>
<td>52</td>
</tr>
<tr>
<td>8.3</td>
<td>Interface</td>
<td>55</td>
</tr>
<tr>
<td>8.4</td>
<td>Iteration</td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td>Pulse program execution</td>
<td>59</td>
</tr>
<tr>
<td>9.1</td>
<td>Device initialization</td>
<td>59</td>
</tr>
<tr>
<td>9.2</td>
<td>Device utilization</td>
<td>59</td>
</tr>
</tbody>
</table>
Part I

Installation
CHAPTER 1

Installation procedure

The latest version available on PyPI (http://pypi.python.org/pypi/SpanishAcquisition/) can be installed using either:

```
sudo pip install SpanishAcquisition
```
or:

```
sudo easy_install SpanishAcquisition
```

Alternatively, this package can be installed from source in the usual way:

```
sudo python setup.py install
```
Dependencies

These are all the dependencies in use by the package.

If only using a subset of the package, it suffices to use what the subset requires. For example, all the GUI-related dependencies are not necessary if only the device-related code is to be used, and vice versa. However, everything is written in (and so requires) Python 2:

- Python (http://www.python.org/) \( \geq 2.6 \land < 3 \)

## 2.1 Device drivers

- NI-VISA (http://www.ni.com/visa/) for Ethernet (and GPIB on Windows) device support.
- Linux GPIB (http://linux-gpib.sourceforge.net/) (with Python bindings) for GPIB device support.

## 2.2 Python packages

- Chaco (http://code.enthought.com/chaco/)
- Enable (http://code.enthought.com/projects/enable/)
- matplotlib (http://matplotlib.sourceforge.net/)
- numpy (http://numpy.scipy.org/)
- ObjectListView (http://objectlistview.sourceforge.net/python/)
- pyparsing (http://pyparsing.wikispaces.com/)
- PyPubSub (http://pubsub.sourceforge.net/) \( \geq 3.1 \)
- PyVISA (http://pyvisa.sourceforge.net/)
- quantities (http://packages.python.org/quantities/)
- scipy (http://www.scipy.org/)
- wxPython (http://www.wxpython.org/)

### 2.2.1 Documentation

- Sphinx (http://sphinx.pocoo.org/) \( \geq 1.0 \)
2.2.2 Testing

- `nose` (http://somethingaboutorange.com/mrl/projects/nose/)
- `nose-testconfig` (http://pypi.python.org/pypi/nose-testconfig/)
Part II

User’s Guide
CHAPTER 3

General concepts

Everything tangible (devices (page 9), variables (page 10)) in this package is tied together via resources (page 9).

3.1 Devices

A device typically refers to a physical piece of hardware, but can be, for example, a piece of software pretending to be hardware.

Devices can offer any number of arbitrary resources (page 9). They may be organized in an hierarchical fashion through subdevices (page 9).

Tip: A device resource will typically correspond to a setting on the device, but it is not required to do so. Other types of resources include measurement readings, and virtual (implementation-specific) settings.

3.1.1 Subdevices

A device which has several versions of the same resources will typically have these resources organized into a tree of subdevices.

For example, an oscilloscope may have multiple input channels, with each having the same set of settings (e.g., vertical scaling and offset). These channels may then be viewed as subdevices, and each will have its own values for all its settings.

See also: Device resources (page 22)

3.1.2 Mock

Mock devices are software simulations of hardware devices. They are typically created at the same time as the device interface and support the same functionality. Most of the time, they are used for testing or as a virtual device.

3.2 Resources

Resources provide a way of interacting with devices (page 9) and pulse programs (page 13) in a generic way. They can be read-only (RO), write-only (WO), or read-write (RW). RO and RW resources are considered readable; WO and RW resources are considered writable.
In the typical case, a device (e.g., a multimeter) will provide several resources; there may be some RW resources (e.g., integration time, auto-zero setting), and some RO ones (e.g., measurement reading). WO resources are possible (and are commonly found as pulse program resources), but are atypical among devices.

These provided resources may be given arbitrary labels, and these labels are used elsewhere to identify the resources. For example, given two RW resources with labels “reading” and “gate”, one can set up a variable which writes to the resource labelled “gate” and a measurement which reads from the resource labelled “reading”.

**Note:** Resources linked to variables must be of the correct type; *output variables* (page 10) shall only be used with writable resources; *input variables* (page 11) shall only be used with readable resources.

Resources may also have associated units, in which case the value read from or written to that resource must be a quantity with a matching dimensionality. That is, a resource which requires acceleration in meters per second squared (specified as \( \text{m.s}^{-2} \)) will accept millimeters per second squared (\( \text{mm.s}^{-2} \)) and joules per newton per second squared (\( \text{J.N}^{-1}.s^{-2} \)), but not joules per newton (\( \text{J.N}^{-1} \)) or joules per second squared (\( \text{J.s}^{-2} \)).

### 3.3 Variables

Variables are used to describe sweeping acquisition experiments.

#### 3.3.1 Output variables

Output variables provide a way to sweep over a range of values on a *resource* (page 9). An output variable will have a customizable range of values over which it iterates.

**Tip:** The unqualified term “variable” typically refers to output variables.

**Type**

An output variable has an associated type. It must be one of:

- **Float** Each produced value is a floating point value, with an integral portion and a decimal portion. For example, “-5.5”.

- **Integer** Each produced value is an integer value. If any of the generated values contain a decimal portion, it is truncated. For example, “5”.

- **Quantity** Each produced value is a floating point value with a corresponding unit. For example, “12.3 GHz”.

**Constant value**

Each variable is assigned a constant value. By default, this value is ignored; however, there are several options which make use of this value.

For example, if the variable is set to be a “constant variable”, then its value is never iterated; instead, its value is set to the constant value at the beginning of a sweep and then left there for the entire duration.

**Note:** The constant value of a variable always incorporates the *type and units* (page 10) of the variable.
See also:

Smooth setting (page 11)

Order

Each output variable has an order to which it belongs, defined by an integer value (either negative, zero, or positive). This order is used to determine the looping sequence for variable iteration and has no bearing on the values of the variable.

Variables which share the same order value are stepped together. Variables which have a greater order value are stepped more slowly (i.e., they are on an outer loop of the iteration) relative to those which have a lesser order value.

**Warning:** In a single order, whichever variable has fewest values dictates how many values the other variables will have. Any excess values for the longer variables are silently truncated.

For example, if variables A, B, C, and D have orders of -5, 1, 1, and 10, respectively, then:

- D will iterate most slowly
- B and C will iterate in lockstep, between D and A
- A will iterate most quickly

Note that since constant variables by definition do not iterate, they are all put into a separate virtual order, and so are ignored from the point of view of the ordering discussion.

Smooth setting

During a sweep, it is sometimes beneficial to avoid abruptly setting variables to values, since this can correspond to large jumps in current or potential difference in configured devices. To reduce the negative effect of changing values, the variables can optionally be “smoothly set” at various times:

- **Smooth setting from constant value** At the start of a sweep, the variable is set to the constant value, and then (over the desired number of steps) swept towards its initial value at the start of the sweep.
- **Smooth setting to constant value** At the end of a sweep (even if the sweep is prematurely aborted), the variable is smoothly swept from its final value to its constant value.
- **Smooth transition between loop iterations** At the end of a single iteration of an order, if that order was not the slowest-stepping outer loop order, the variable is smoothly swept back to its initial value so that it can be stepped over again.

Note: Each smooth setting step is always 100 ms in duration.

See also:

Variable configuration (page 30)

### 3.3.2 Input variables

Typically referred to as “measurements”, input variables provide a way of gathering data from resources (page 9).

There exist two types of measurements: scalar and list. Scalar measurements correspond to the acquisition of single values over time (e.g., an amplitude or a frequency); list measurements correspond to the acquisition of a list of values.
over time (e.g., a waveform captured by an oscilloscope). Naturally, if the measurements are done several times, scalar measurements produce one-dimensional data, while list measurements produce two-dimensional data.

See also:

*Measurement configuration* (page 22)
Pulse programs are an unambiguous way to describe the parameterizable shape of one or more parallel waveforms. Pulse programs allow for:

- **Adjustable time scales.** Waveforms can be on the order of nanoseconds, of seconds, or anything in between.
- **Arbitrary parameterization.** Delays, pulse amplitudes and length, and repetition counts can all be iterated over.
- **Arbitrary waveforms.** Any discretized waveform can be inserted.
- **External trigger.** An oscilloscope acquisition can be triggered at any point during the waveforms.
- **Nested looping.** Complex, repetitive waveforms can be created easily.

### 4.1 Variables

In order to allow parameterization, any values which can be directly used (such as integers or delays) can alternatively be used with a variable. Some values (such as pulses and outputs) require the use of a variable.

#### 4.1.1 Types

Pulse programs can contain the following variable types:

- **Integer (int)** A single integer value, such as 12 or -5.
- **Delay (delay)** A single time quantity, such as 5.5 ns.
- **Pulse (pulse)** A collection of three values set using a dictionary or attributes:
  - **amplitude**: A voltage quantity, such as 250 mV.
  - **length**: A time quantity.
  - **shape**: A string containing a valid file name.
- **Output (output)** A special type which does not support assignment. Outputs are always configured when the pulse program is to be used.
4.1.2 Dictionaries

Any variables which are collections of values can have several of their values assigned via a dictionary. Dictionaries are of the form \{<key>: <value>, \ldots\} where the keys must correspond to the attribute names of the variable being assigned to. For example:

# A valid dictionary assigned to a pulse:
p1 = \{amplitude: -0.5 V, length: 0.5 us, shape: 'square'\}

# Not all attributes need be present:
p2 = \{shape: 'non-square'\}

4.1.3 Attributes

Any variables which are collections of values can have a single value assigned via an attribute. For example:

# Assignment to a pulse attribute:
p3.amplitude = -500 mV

4.2 Program syntax

A pulse program consists of several statements separated by line breaks or semicolons (;). A statement is one of:

**Assignment** An assignment sets the value of a variable or attribute. Any variable or attribute may only be assigned to once in a single program. For example:

p1 = \{shape: 'square', length: 1 ns, amplitude: 1 V\}
dl = 100 ns
loops = 5

**Declaration** A declaration is an announcement of intent to use a variable; all used variables must be declared at some point in the program. A declaration consists of a type, followed by one or more identifiers; any identifier may also be part of an assignment within the declaration. For example:

int bumps = 2
delay bump_spacing = 20 ns, settle, end_delay

**Command** A command is an instruction that dictates the shape of the resulting waveforms. There exist several kinds of commands:

- **Delay**: A lone identifier or a single time value causes a delay in all output waveforms. For example:

  # Pause all waveforms for the length of delay dl:
dl

  # Pause all waveforms for 100 ns:
dl

  100 ns

- **Pulse sequence**: Statements of the form \{([delay|pulse] [delay|pulse] \ldots\):<output>\} are treated as waveform-generating command. If there is only a single delay or pulse, the parentheses may be omitted. If several pulse sequences are included in the same statement, they are executed in parallel; otherwise they are executed in series. For example:
# Generate delay or pulse x on output f1:
x:f1
# Alternatively:
(x):f1

# Generate delay or pulse x, followed by a 10 ns delay, followed by
# another instance of delay or pulse x, all in sequence on output f2:
(x 10 ns x):f2

# Generate identical simultaneous waveforms on outputs f1 and f2:
(x d1 y):f1 (x d1 y):f2

# Generate different simultaneous waveforms on outputs f1, f2, and f3:
x:f1 (x d1 y):f2 y:f3

# Same output shapes as above, but the waveforms on the different outputs
# follow one another in time:
x:f1
(x y):f2
y:f3
# Alternatively:
x:f1 ; (x y):f2 ; y:f3

Note: All waveforms are synchronized before and after a pulse sequence. If any pulse sequence
would be longer than the others, padding delays are automatically added to the end of the shorter
sequences to ensure that all the lengths match.

• Acquisition trigger: A statement of the form acquire signals that an oscilloscope acquisi-
tion trigger must occur on an output at that point. Such triggers are always created on output
markers, rather than as part of the output waveform itself.

Loop A loop is a section of the program which is to be executed several times. The contents of a loop
block are constrained to delays, pulse sequences, and loops. Loops are of the form:

```
times <integer> {
    <statement>
    ...
}
```

4.2.1 Comments

Any text after (and including) a # character is ignored. For example:

```
# This is a pulse sequence.
(p1 d1 p1):f1 # (p2 d2 p2):f2
```

is identical to:

```
(p1 d1 p1):f1
```

4.3 Parameterization

Any values which are not assigned in the body of the pulse program must be filled in at a later time. For example:
pulse p1 = {amplitude: 1 V, shape: ‘square’}
output f1

p1:f1

is the entirety of valid pulse program, but p1.length is treated as an external parameter and must be known in order to generate the waveform for output f1.

It is possible to fill these values in dynamically as part of a sweep, given that the parameters are assigned resource labels.

See also:

Pulse program configuration (page 26)

4.4 Examples

The following examples all use a sampling rate of 1 GHz.

4.4.1 Single waveform

delay d1 = 5 ns
int bumps
pulse p1 = {amplitude: 1 V, shape: ‘square’}
output f1

p1.length = 10 ns

3 ns
p1:f1
times bumps {
    d1
    (p1 1 ns p1):f1
}

If the parameter bumps is filled in with the value 3, the following waveform is generated:

4.4.2 Multiple waveforms

pulse p1 = {amplitude: 0.5 V, length: 10 ns, shape: ‘non-square’}
pulse p2 = {amplitude: -1.5 V, length: 5 ns, shape: ‘non-square’}
output f1, f2

1 ns
p1:f1
1 ns
(p1 2 ns p1):f1 (p2 3 ns p2):f2
5 ns
p2:f2
8 ns

If the file “non-square” contains the data “-0.1, 0.0, 0.1, 0.2, 0.4, 0.8, 1.6”, the following pair of waveforms is generated:
4.4.3 With acquisition

pulse p1 = {amplitude: 0.25 V, length: 15 ns, shape: 'square'}
output markered

20 ns
p1:markered
acquire
p1:markered
20 ns

If the acquisition marker is set up to be marker 2 on output markered, the following output waveform and marker waveform are generated:
CHAPTER 5

Graphical user interface

There are several GUI elements which work together, and can be assembled arbitrarily into graphical applications.

5.1 Action

5.1.1 Data capture

Data capture panel

The data capture panel controls the verification of the setup, as well as the export of the acquired data.

1. Begin the sweep as defined in the rest of the application. If there are any setup errors, pressing this button will generate messages about the errors rather than beginning the sweep.
2. If the “Continuous” checkbox is enabled, the sweep will restart from the beginning as soon as it gets to the end.
3. If the “Export” checkbox is enabled, the sweep values (of both input and output variables) will be exported; otherwise, they will be discarded.
4. The location of the directory to which the values should be exported.
5. The location of the file to which the last set of values was exported.

Data capture dialog

The data capture dialog controls the sweep itself.

The sweep consists of the following stages:

- Initializing Internal setup.
- Getting next values The next set of values to write to any resources is determined based on the variable configuration.
- Smooth setting A smooth transition occurs for all resources requiring one.
- Writing to devices All variables with changed values have those values written to their resources.
1. The overall progress of the sweep.
2. The currently executing stage of the sweep.
3. The last values set to the output variables.
4. The last values obtained from the measurements.
5. The total elapsed time for the sweep. In non-continuous mode, this is accompanied by a rough estimate for the remaining time.
6. Checkbox controlling the termination of a sweep in continuous mode. At the end of a sweep in continuous mode, if this checkbox is enabled, the sweep terminates; otherwise, it restarts.
7. Prematurely terminate a sweep.

Tip: To avoid leaving the system in an inconsistent state, pressing the “Cancel” button first allows whichever stage is currently running to finish running gracefully. Then, if any variables were configured to be set smoothly from their final values to their constant values, they are set smoothly from wherever the sweep was ended. Thus, it is safe to cancel the sweep at any time.
**Waiting for resources to settle** A delay occurs for the variable with the longest wait time.

**Running pulse program** If one is configured, a pulse program is run.

**Taking measurements** All measurements are read from their resources.

**Smooth setting** A smooth transition occurs from the last value to the variables’ constant values, as required.

**Finishing** Internal cleanup.

**Export format**

The export is done to a regular comma-separated values (CSV) file. The first row contains the column headings (potentially with units) as gathered from the variable and measurement names. The first column is always titled “Time (s)” and contains the approximate time of acquisition for each row, relative to the first row of values.

For example, an exported file may begin thusly:

```
Time (s),field (mT),port out (V),Pulses (V),port in (V)
0,1.0,-5.0,[(0.0, -0.078469520103761514), ...],-0.0100000004
0.25141787529,1.0,-4.375,[(0.0, -0.11684596017395288), ...],-0.0100000008
```

Note that the “Pulses” column contains list data and most of it has been elided for clarity.

**5.1.2 Smooth reset**

The smooth reset panel provides a convenient way to reset all resources tied to enabled and constant variables. Each step is always 100 ms.

1. Sweep from constant values to zero.
2. Sweep from zero to constant values.
3. The number of steps to use.

**Warning:** For both directions, the smooth reset panel makes the assumption that the from value is the current value of the resource. For example, if the voltage source port behind the resource is sitting at 5 V, and the constant value of the variable is 5 V, pressing the “To zero” button is safe. If this is not the case, a jump will occur at the beginning of the reset.

**5.2 Configuration**

**5.2.1 Device configuration**

**Device list**

The device list shows all the devices (page 9) configured in an application.
1. A unique label to identify the device.
2. The connection status. Double-clicking in this column brings up the device configuration dialog for the device.
3. Graphical configuration. If a device has this option, “Setup...” appears in this column; double-clicking it opens up the device-specific graphical configuration.
4. The status of any long-running tasks.
5. Clicking “Add” creates a blank device. Clicking “Remove” permanently removes all selected devices.

**Device configuration dialog**

The device configuration dialog is used to configure an individual *device* (page 9) for use with an application.

**Device connection**

Connection and model setup.

**Note:** Disconnecting from a device does not actually take place until the dialog is confirmed (by pressing “OK”). Thus, pressing “Disconnect” and then “Cancel” will retain the connection to the device.

**Device resources**

Resource labels and values setup.

**5.2.2 Measurement configuration**

The measurement configuration dialogs are used to set up *input variables* (page 11) for an application.

**Note:** If the plots cannot be displayed (most likely due to a required plotting library not being installed), the plots and all plot-specific options will be omitted. However, the measurements can still be configured for acquisition.

**Scalar measurement**

A scalar measurement consists of a single data point for each point in time.
1. Address configuration.
   - Ethernet: The IPv4 address (eg. 1.2.3.4).
   - GPIB: The board number, PAD, and SAD.
   - USB: The full USB resource (eg. USB::0x1234::0x5678::01234567::RAW).

2. Implementation configuration.
   - The manufacturer and model must be selected.
   - When “Mock” is selected, a software mock implementation (page 9) is used instead of connecting to a real device.

3. Connection control.
   - Connect to or disconnect from the device.

4. Saving settings.
   - All the settings controlled in this dialog (including resource labels, but with the exception of resource values) can be saved to and loaded from the disk.
1. The internal name of the resource.

2. R/W flags indicating the RO/WO/RW status of the resource.

3. The units associated with the resource.

4. The unique label used to identify the resource. The label can be changed by double-clicking the appropriate field.
   In this case, the channel 1 waveform has been given the label osc\textbf{1}.

5. The value of the resource.
   - If the resource is readable, the latest received value is displayed. The exception to this is slow resources, which always have “[N/A]” displayed to avoid slowdowns caused by fetching the value.
   - If the resource is writable, the value can be changed by double-clicking the appropriate field.

6. A subdevice with its own resources.
1. Whether the measurement should be used during a sweep.
2. The label of the resource from which this measurement should acquire data.
3. A unique label to identify the measurement. This name appears, for example, as a column heading when capturing data.
5. Historical view of the measurement.
6. The last value read from the resource.
7. “Run” and “Pause” control the live-view when there is no sweep in progress. “Reset” clears the historical view.
8. Plot settings (page 26) dialog.
Resource scaling

Resources can be scaled as desired after they come in from a device, but before they are used.

The resource used for the measurement is scaled according to the formula:

\[ \text{new value} = (\text{linear scale}) \times (\text{old value}) \times 10^{\text{exponential scale}} + \text{offset} \]

**Note:** This scaling affects the *recorded* and *exported* values as well as the displayed values.

Plot settings

The historical plot can be configured, or disabled outright.

List measurement

A list measurement consists of a list of data point for each point in time.

The list measurement configuration is identical to scalar measurement configuration (page 22), but with fewer options.

Plot settings

The historical plot can be configured, or disabled outright.

5.2.3 Pulse program configuration

The pulse program configuration panel is used to set up the *pulse program* (page 13) for an application.

If a pulse program is loaded (via File -> Open... or Ctrl+O), the sweep performed by the *Data capture dialog* (page 19) will include a stage involving the pulse program. If it is later closed (via File -> Close or Ctrl+W) or if one was not opened, the corresponding sweep stage is skipped.

**Note:** To run a pulse program, it is necessary to have configured both an AWG (to emit the waveforms) and an oscilloscope (to take measurements).

Resources

Most parameterizable values of a pulse program may be given *resource labels* (page 9). These resource labels behave the same way as device resources, and so may be iterated over by a variable during a sweep.
• **Enabled**: Whether the plot should be updated at all.

• **Capture**
  - **Points**: The number of historical values to display. The maximum value is 10,000.
  - **Delay**: In live-view mode, the delay between successive acquisitions. The minimum value is 200 ms.

• **Axes**
  - **x**
    - **Autofit**: Whether to automatically re-scale the x axis as values change.
    - **Value**: The unit (time or data point number) along the x axis.
    - **Mode**: Whether to use relative or absolute values.
  - **y**
    - **Autofit**: Whether to automatically re-scale the y axis as values change.
    - **Conversion**
      - **Exp. scale**: The incoming value is scaled according to the formula:
        \[ \text{new value} = \text{old value} \times 10^{\text{exp scale}} \]
      - **Units**: The incoming units are assumed to be the “From” units and the displayed value is adjusted to match the “To” units.

**Note**: These conversion settings affect only the displayed values.
- Enabled: Whether the plot should be updated at all.
- Capture
  - Lines: The number of lines of historical data to display.
Setup

Tip: Each setup tab is present only if the pulse program requires it. For example, the Acquisition tab appears only if the command `acquire` is in the program.

Acquisition

Configuration for the acquisition trigger.

<table>
<thead>
<tr>
<th>Acquisition</th>
<th>Delays</th>
<th>Integers</th>
<th>Outputs</th>
<th>Pulses</th>
</tr>
</thead>
<tbody>
<tr>
<td>marker_num</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>f1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Times to average</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-trigger delay</td>
<td>100 us</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- marker_num: The number of the marker of the AWG channel output used for the trigger.
- output: The waveform output name.
- Times to average: If this item is set to a value greater than 1, the oscilloscope is placed into “average” mode for the given number of waveforms, and the AWG is triggered that many times.
- Post-trigger delay: How long to wait after triggering the AWG. This delay occurs between individual waveforms when averaging.

Delays, Integers, Pulses

Configuration for the pulse program variables.

Delays, integers, and pulses are all configured in approximately the same manner, and most of their aspects can be parameterized.

All items (for delays and integers, the variables themselves; for pulses, the variable `attributes` (page 14)) appear exactly as they are named in the pulse program. Each must have a default value, and may also have an optional resource label.

The default value is used when displaying the output waveform in the Outputs tab, and when sending the waveform to the AWG given that either no resource label is provided or no value is written to the resource during the sweep.

Outputs

Configuration for the waveform outputs.
5.2.4 Variable configuration

Variable configuration panel

The variable configuration panel is used to set up output variables (page 10) for an application.

To select a variable, click on its row. To select multiple variables, hold down the “ctrl” key while clicking. When several variables are selected, some actions (such as clicking “Remove” or pressing the space bar) act on all of them.

Tip: The user interface is organized so that gate 2 and gate 3 are still displayed alongside the other variables with order number 1. However, gate 2 is set to const, and so will be in a separate virtual order, and gate 3 is disabled, so will not participate at all.

Variable sweep example

The configuration shown above (page 31) would result in the following actions during a sweep:

1. Resource v1 is smoothly stepped from -2.5 V to -5 V.
2. The values are set (with each being written to the appropriate resource, if any):
1. The “enabled” checkbox. Variables which do not have this checked (such as gate 3) effectively do not exist; this is useful for temporarily disabling variables.

2. A unique label to identify the variable. This name appears, for example, as a column heading when capturing data.

3. The order number of variables is used to group them during the sweep. gate 1 and gate 4 have the same order, so would be stepped together in the inner loop; magnetic field has a higher order number, and so will be stepped alone in the outer loop.

4. The resource label for the resource to which to write the values. All the resources provided must be writable. If a resource is not provided (such as with gate 4), the variable is still stepped in the usual fashion, but its values are discarded.

5. The values over which the variable will be stepped. If there are too many values, some are omitted from the display. The symbols on either side of the values specify whether that side is set smoothly: “(“ and “)“ if smoothly (as for gate 1); “[“ and “]” if not (as for the other variables).

If there are any units associated with a variable, they are displayed after the values.

6. For each step of a variable, after writing the value to the resource, there is a delay of at least the wait time. In each order, the delay for all variables is the longest of the wait times in that order. The effective wait time for gate 4 is 200 ms.

7. The “const” checkbox. Variables which have this checked (such as gate 2) are considered constant and are subject to special consideration in some scenarios.

8. The const value of a variable is that which it is assumed to take on at rest. The value does nothing on its own, but is used in conjunction with other settings and actions.

The units associated with a variable apply to the const value as well.

9. Clicking “Add” creates a blank variable. Clicking “Remove” permanently removes all selected variables.

10. The variable settings can be saved to and loaded from the disk. All the configured variables (both enabled and not) are saved at the same time, and existing variables are overwritten by any loaded variables.

<table>
<thead>
<tr>
<th>Name</th>
<th>Resource</th>
<th>Values</th>
<th>Wait time</th>
<th>Const. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gate 1</td>
<td>v1</td>
<td>[-5., -4.375, -3.75, -3.125, ..., 0] V</td>
<td>200 ms</td>
<td>-2.5</td>
</tr>
<tr>
<td>gate 2</td>
<td>v2</td>
<td>[5.6] V</td>
<td>200 ms</td>
<td>5.6</td>
</tr>
<tr>
<td>gate 3</td>
<td>v3</td>
<td>[0]</td>
<td>0 s</td>
<td>-5.6</td>
</tr>
<tr>
<td>gate 4</td>
<td></td>
<td>[-5., -3.75, -2.5, -1.25, ..., 5] V</td>
<td>150 ms</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>magnetic field</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Between steps of variables in order 2, if gate 1 set to smoothly transition, resource v1 is smoothly stepped from 0.0 V to -5.0 V.

3. Resource v1 is smoothly stepped from 0 V to -2.5 V.

**Variable editor dialog**

The variable editor dialog is used to configure the values over which a variable is stepped. It is opened by double-clicking in the “Values” column of the variable.

1. The value configuration is performed by using one of the available configuration panels.
2. Smooth setting (page 11) configuration.
3. Type and units (page 10) configuration.
Configuration panels

Linear A linear space is described between the initial and final bounds (inclusive), consisting of the specified number of values. For example, if initial, final, and steps are were to 1, 5, and 9, respectively, the resulting values would be: 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5.

Arbitrary Values are provided directly as a sequence of comma-separated numbers (with ignored whitespace). For example, the input “1, 32, -5, 6.543, 0, 0, 1” would result in the values: 1, 32, -5, 6.543, 0, 0, 1.

5.3 Display

5.3.1 Plots

Via other libraries, this package provides several ways of visualizing data though plots.

Colormapped plot

The colormapped plot display allows for the display of one dimension in the color space versus two other dimensions in the plane.

![Colormapped plot](image)

Pan & zoom

In addition to zooming within the plane, the colormapped plot allows for zooming in the color space. Changing the values in the “Min” and “Max” fields, followed by pressing the enter key (once for each changed value) will change the bounds of the color space (and consequently of the legend).

See also:

Pan & zoom (page 36)

In order to highlight a particular value region, one can also click the right mouse button on the legend and drag along the legend. This will select the region to be highlighted; everything outside this region will be dimmed:
Surface plot

The colormapped plot display allows for the display of one dimension in the vertical direction versus two other dimensions in the plane.

Pan & zoom

Clicking anywhere on the plot with the left mouse button and dragging in any direction will cause the axes and the surface to rotate in the direction of the drag.

Clicking anywhere on the plot with the right mouse button and dragging up or down will cause the surface to zoom in and out.
Waveform style

The surface plot supports an alternate style suited for the display of discrete waveforms rather than of a two-dimensional sweep.

Two-dimensional plot

The two-dimensional plot allows for the display of one dimension versus another dimension, both in a single plane.
Pan & zoom

Clicking anywhere on the plot with the left mouse button and dragging in any direction will cause the axes and the plot to pan in the direction of the drag.

Scrolling the scroll wheel of the mouse will cause the plot to zoom in or out, depending on the direction of the scroll. To only zoom in a single direction at a time, one can hold the shift key while pressing the arrow keys; up/down control zoom in the vertical direction (in/out, respectively), while right/left control zoom in the horizontal direction (in/out, respectively).

To get a more precise zoom, one can hold the control key while dragging with the left mouse button to draw an exact selection rectangle. Releasing the left mouse button will cause the selected area to take up the entire visible area.

Pressing the escape key will reset the zoom (but not the pan) to the original settings.

5.3.2 Tabular display

The tabular display panel is a read-only panel for arbitrary data.

<table>
<thead>
<tr>
<th>time [s]</th>
<th>field [mT]</th>
<th>port out [V]</th>
<th>Pulses [V]</th>
<th>port in [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.024776</td>
<td>1.0</td>
<td>-2.5</td>
<td>(0.0, 0.097695887693598937), (1.00100... -0.0100000064</td>
<td></td>
</tr>
<tr>
<td>1.365668</td>
<td>1.0</td>
<td>-1.875</td>
<td>(0.0, -0.057619593584111545), (1.00100... -0.0100000128</td>
<td></td>
</tr>
<tr>
<td>1.605982</td>
<td>1.0</td>
<td>-1.25</td>
<td>(0.0, 0.11768520637827118), (1.00100... -0.0100000256</td>
<td></td>
</tr>
<tr>
<td>1.876083</td>
<td>1.0</td>
<td>-0.625</td>
<td>(0.0, -0.077782864138410178), (1.00100... -0.0100000512</td>
<td></td>
</tr>
<tr>
<td>2.145252</td>
<td>1.0</td>
<td>0.0</td>
<td>(0.0, 0.0393350128938734897), (1.00100... -0.0100001024</td>
<td></td>
</tr>
<tr>
<td>4.211317</td>
<td>3.0</td>
<td>-5.0</td>
<td>(0.0, -0.13660639353017467), (1.00100... -0.0100002048</td>
<td></td>
</tr>
<tr>
<td>4.523829</td>
<td>3.0</td>
<td>-4.375</td>
<td>(0.0, -0.059014267185473201), (1.00100... -0.0100004096</td>
<td></td>
</tr>
<tr>
<td>4.788892</td>
<td>3.0</td>
<td>-3.75</td>
<td>(0.0, -0.078088044556344105), (1.00100... -0.0100008192</td>
<td></td>
</tr>
<tr>
<td>5.056252</td>
<td>3.0</td>
<td>-3.125</td>
<td>(0.0, 0.019569695582513091), (1.00100... -0.0100016384</td>
<td></td>
</tr>
</tbody>
</table>

Column types

The panel is able to recognize certain data types: scalars, lists, and strings. Scalars are any real numbers, lists are number sequences with a particular syntax, and everything else is considered to be a string.

Lists are of the form:

\[
[(\text{number, number}), (\text{number, number}), \ldots]
\]

This is the format used by the Data capture (page 19) panel for export of list data.

Data filters

Data filters allow the user to narrow a large dataset into a smaller one, potentially making visualization simpler and all operations faster.

Tip: If there are already filters which have transformed an originally large dataset into a small one, adding a new filter is a very fast operation. However, removing or editing any filter will be as slow as applying the first filter, since the original dataset must be re-filtered in its entirety.
Syntax

Filters are provided as Python-like expressions on a per-column basis. All occurrences of $x$ are internally replaced with an identifier for the selected column, and the remainder of the expression is left intact. Thus, the user is free to use as much creativity as desired when constructing filters.

Valid comparison operators include: $<$ (less than), $<=$ (less than or equal to), $==$ (equal to), $!=$ (not equal to), $>=$ (greater than or equal to), $>$ (greater than). Comparisons may be grouped with: and (both expressions must be true), or (at least one expression must be true); and negated with: not (expression must be false).

Tip: The Python equality comparison operator is two equal signs ($==$), not a single equal sign ($=$). Using the latter in place of the former may generate a generic “invalid syntax” error.

Examples

All examples assume the following initial dataset:

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>field (mT)</th>
<th>port out (V)</th>
<th>port in (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
<td>-5.0</td>
<td>-0.0100000004</td>
</tr>
<tr>
<td>0.251417</td>
<td>1.0</td>
<td>-4.0</td>
<td>-0.0100000008</td>
</tr>
<tr>
<td>0.512650</td>
<td>1.0</td>
<td>-3.0</td>
<td>-0.0100000016</td>
</tr>
<tr>
<td>0.766408</td>
<td>1.0</td>
<td>-2.0</td>
<td>-0.0100000032</td>
</tr>
<tr>
<td>1.024776</td>
<td>1.0</td>
<td>-1.0</td>
<td>-0.0100000064</td>
</tr>
<tr>
<td>1.300688</td>
<td>3.0</td>
<td>-5.0</td>
<td>-0.0100000128</td>
</tr>
<tr>
<td>1.605982</td>
<td>3.0</td>
<td>-4.0</td>
<td>-0.0100000256</td>
</tr>
<tr>
<td>1.876083</td>
<td>3.0</td>
<td>-3.0</td>
<td>-0.0100000512</td>
</tr>
<tr>
<td>2.145252</td>
<td>3.0</td>
<td>-2.0</td>
<td>-0.0100001024</td>
</tr>
<tr>
<td>4.211317</td>
<td>3.0</td>
<td>-1.0</td>
<td>-0.0100002048</td>
</tr>
<tr>
<td>4.523829</td>
<td>5.0</td>
<td>-5.0</td>
<td>-0.0100004096</td>
</tr>
<tr>
<td>4.788892</td>
<td>5.0</td>
<td>-4.0</td>
<td>-0.0100008192</td>
</tr>
<tr>
<td>5.056252</td>
<td>5.0</td>
<td>-3.0</td>
<td>-0.0100016384</td>
</tr>
<tr>
<td>5.353702</td>
<td>5.0</td>
<td>-2.0</td>
<td>-0.0100032768</td>
</tr>
<tr>
<td>5.627074</td>
<td>5.0</td>
<td>-1.0</td>
<td>-0.0100065536</td>
</tr>
</tbody>
</table>

**Time selection**  Select only the rows that fit within a slice of time.

The filter:

<table>
<thead>
<tr>
<th>Column:</th>
<th>Time (s)</th>
<th>x &gt; 1.0 and x &lt; 5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

results in:

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>field (mT)</th>
<th>port out (V)</th>
<th>port in (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.024776</td>
<td>1.0</td>
<td>-1.0</td>
<td>-0.0100000064</td>
</tr>
<tr>
<td>1.300688</td>
<td>3.0</td>
<td>-5.0</td>
<td>-0.0100000128</td>
</tr>
<tr>
<td>1.605982</td>
<td>3.0</td>
<td>-4.0</td>
<td>-0.0100000256</td>
</tr>
<tr>
<td>1.876083</td>
<td>3.0</td>
<td>-3.0</td>
<td>-0.0100000512</td>
</tr>
<tr>
<td>2.145252</td>
<td>3.0</td>
<td>-2.0</td>
<td>-0.0100001024</td>
</tr>
<tr>
<td>4.211317</td>
<td>3.0</td>
<td>-1.0</td>
<td>-0.0100002048</td>
</tr>
<tr>
<td>4.523829</td>
<td>5.0</td>
<td>-5.0</td>
<td>-0.0100004096</td>
</tr>
<tr>
<td>4.788892</td>
<td>5.0</td>
<td>-4.0</td>
<td>-0.0100008192</td>
</tr>
</tbody>
</table>
**Dimensionality reduction**  Reduce an entire dimension to a single point.

This allows for plots which would otherwise be impossible. For example, “port in (V)” vs “port out (V)” cannot be plotted since “port in (V)” is not a function of “port out (V)”.

The filter:

<table>
<thead>
<tr>
<th>Column:</th>
<th>field (mT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>x == 3.0</td>
</tr>
</tbody>
</table>

results in:

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>field (mT)</th>
<th>port out (V)</th>
<th>port in (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.300688</td>
<td>3.0</td>
<td>-5.0</td>
<td>-0.0100000128</td>
</tr>
<tr>
<td>1.605982</td>
<td>3.0</td>
<td>-4.0</td>
<td>-0.0100000256</td>
</tr>
<tr>
<td>1.876083</td>
<td>3.0</td>
<td>-3.0</td>
<td>-0.0100000512</td>
</tr>
<tr>
<td>2.145252</td>
<td>3.0</td>
<td>-2.0</td>
<td>-0.0100001024</td>
</tr>
<tr>
<td>4.211317</td>
<td>3.0</td>
<td>-1.0</td>
<td>-0.0100002048</td>
</tr>
</tbody>
</table>

It becomes possible to plot “port in (V)” vs “port out (V)” at a constant “field (mT)”.

**Configuration**

**Filter list**  The filter list displays all existing filters.

Filters can be added with the “Add” button, permanently removed with the “Remove” button, and edited by double-clicking on the respective row.

**Filter editor**  The filter editor allows the user to create new filters and to edit existing filters.

The given filter is added to the selected column.

If there is an error in the input, the user is informed. For example, the filter “y == 5” results in the message “name ‘y’ is not defined”.

38 Chapter 5. Graphical user interface
Example applications

Spanish Acquisition comes bundled with example applications, showing its use and capabilities. The examples are found in examples/. To run an example if Spanish Acquisition is installed (page 3), simply run the example Python file; otherwise, use python -m examples.<name>.

6.1 Acquisition

The Acquisition application allows the user to: set up multi-dimensional sweeps (page 10) of resources (page 9) on physical devices (page 9), acquire the resulting values, and run pulse programs (page 13).

The Acquisition application is composed of several interacting components:

- In the main window:
  - Top: Variable configuration (page 30)
  - Bottom-left: Data capture (page 19)
  - Bottom-right: Smooth reset (page 21)

- Accessible through the menus:
  - Configuration -> Devices...: Device configuration (page 21)
  - Configuration -> Measurements: Measurement configuration (page 22)
  - Configuration -> Pulse program...: Pulse program configuration (page 26)

6.2 Data Explorer

The Data Explorer is an application that allows the user to view arbitrary CSV files, with the option to display some values in graphical form.

The data explorer is composed of a Tabular display (page 36) panel, along with a menu to load and plot data. It supports data filters (page 36), which can be accessed via File -> Filters....

6.2.1 Plotting options

Combinations of scalar columns can be plotted in two or three dimensions, with the “Curve...”, “Colormapped...”, and “Surface...” menu options. These correspond to the Two-dimensional plot (page 35), Colormapped plot (page 33), and Surface plot (page 34).
List columns can be plotted in three dimensions with the “Waveforms...” menu option, which corresponds to the *Surface plot* (page 34) in *Waveform style* (page 35).
Part III

Developer’s Guide
General information

The developer is assumed to have read through the *User’s Guide* (page 9) before proceeding.

### 7.1 Testing

All the non-GUI modules in this package have associated tests, found in the *tests/* subdirectory. These are split into unit tests (found directly in the subdirectory) and server tests (found in *tests/server/*); server tests differ from unit tests by the addition of an external dependency, such as a hardware device.

**Note:** Test failures can signify a missing *dependency* (page 3). While users can get by with only a subset, it is recommended that developers install *all* dependencies.

#### 7.1.1 Unit tests

The unit tests can all be run with:

```
./runtests
```

#### 7.1.2 Server tests

Currently, only device tests exist in the form of server tests. Before running these, the devices should be connected to the computer and *configured* (page 49).

Server tests can be found with:

```
find ./spacq/ -path '*/tests/server/test_*.py'
```

and run with, for example:

```
./runtests --no-skip ./spacq/devices/tests/server/test_abstract_device.py
```
7.2 Documentation

7.2.1 Style

An attempt has been made to ensure that the reStructuredText source for this documentation is self-consistent. The following stylistic choices have been made:

- The headings *for each document* are as follows:

```
###
One
###
Two
***
Three
=====
Four
----
```

Headings nested deeper than this should not be used.

- A single blank line separates elements within a document. Multiple blank lines are not used.

7.2.2 Building

The documentation can be built using:

```
make -C docs html
```

This places the result at `docs/_build/html/index.html`.

7.2.3 Updating

It is crucial that all modifications which cause any changes to the user interface, whether they are changes in the GUI or changes in behaviour, are documented in the *User’s Guide* (page 9) as they occur. All changes (especially internal ones not mentioned in the *User’s Guide* (page 9)) should be documented in the *Developer’s Guide* (page 45).

7.3 Example applications

The example applications are provided to demonstrate the functionality that this package provides in the form of directly usable and useful applications. As this functionality changes, the example programs should follow suit.

7.4 Creating a release

When a new version is ready for release, the version number must be updated in the following places:

- `docs/conf.py`: `version, release`
- `spacq/__init__.py`: `VERSION`
• setup.py: version

The CHANGELOG.rst file must be updated with all the important changes from the previous version.

To create a source distribution, run:

```
python setup.py sdist
```
8.1 Devices

Interfaces for various hardware devices. This package allows the remaining components to talk to devices through a layer of abstraction.

8.1.1 Class hierarchy

All device classes inherit from `spacq.devices.abstract_device.AbstractDevice`, which provides all the functionality necessary to make connections to physical devices. Subdevices inherit from `spacq.devices.abstract_device.AbstractSubdevice`. Each device class has its own test class, as well as a mock implementation.

`spacq.devices.config.DeviceConfig` and `spacq.devices.config.device_tree()` provide a way to find existing devices and connect to them.

8.1.2 Device implementation notes

**Tektronix AWG5014B**

The implementation for the Tektronix AWG5014B arbitrary waveform generator is `spacq.devices.tektronix.awg5014b.AWG5014B`.

The following examples assume `awg = AWG5014B(...)`. 

**Run modes**

The currently-supported run modes are “continuous” (`awg.run_mode = 'continuous'`) and “triggered” (`awg.run_mode = 'triggered'`). In “continuous” mode, the device will output all the waveforms on the enabled channels until stopped. In “triggered” mode, the device will output all the waveforms on the enabled channels once each time it is triggered (`awg.trigger()`).

**Tektronix DPO7104**

The implementation for the Tektronix DPO7104 oscilloscope is `spacq.devices.tektronix.dpo7104.DPO7104`.

The following examples assume `dpo = DPO7104(...)`. 

“FastFrame” mode

In order to allow fast triggering for on-board averaging, the device can be placed into “FastFrame” mode (dpo.fastframe = True), which enables memory segmentation. The acquisition memory is split up into the requested number of frames (dpo.fastframe_count = 17); this has the negative side effect of decreasing the maximum allowed resolution, since all frames share the same acquisition memory.

The final frame must be enabled in either average (dpo.fastframe_sum = ‘average’) or envelope (dpo.fastframe_sum = ‘envelope’) mode.

Note: dpo.fastframe_count includes the final frame.

Waveform acquisition

The property spacq.devices.tektronix.dpo7104.Channel.waveform downloads a single frame from the device. If “FastFrame” mode is enabled (regardless of the final frame setting), only the last frame is downloaded.

Extremely large waveforms are downloaded in chunks (the size of which is specified by dpo.max_receive_samples) and are assembled into a single waveform locally.

8.1.3 Testing

All the device interfaces can be tested against real hardware as long as the hardware is present and configured.

Configuration of external resources should be done by copying and editing the example file:

```
cp test-config.py ~/.spacq-test-config.py
```

8.1.4 File structure

Each manufacturer has its own directory in spacq/devices/ (eg. agilent for Agilent, tektronix for Tektronix, etc), and each of these directories may contain any number of devices. There should be at least four files per device:

1. The device interface (eg. dm34410a.py).
2. Tests for the device interface (eg. test_dm34410a.py).
3. A mock implementation of the device (eg. mock_dm34410a.py).
4. A wrapper to run the device tests against the mock (eg. test_mock_dm34410a.py).

Adding a manufacturer

To add a manufacturer:

1. Copy the sample manufacturer directory (spacq/devices/sample/) to a new directory in spacq/devices/ corresponding to the manufacturer. The name of this directory will be the package name; it should include only lowercase letters, but starting with the second character may also include digits and underscores.

---

1 Most devices will have four files, but, for example, the IQC voltage source (iqc/voltage_source.py) includes a non-server test file (iqc/tests/test_voltage_source.py) as a fifth file, and a graphical configuration panel (page 51) (iqc/gui/voltage_source.py) as a sixth file.
2. In the spacq/devices/<manufacturer>/__init__.py file, replace the name with the name of the manufacturer as you would like it to appear in the user interface.

3. Add the new package name to both lines of spacq/devices/__init__.py. For example, to add Oxford Instruments as a manufacturer:

cp -r spacq/devices/sample spacq/devices/oxford
vim spacq/devices/oxford/__init__.py # Change the name to "Oxford Instruments".
vim spacq/devices/__init__.py # Add "oxford".

Adding a device

The sample manufacturer comes with a sample device in the form of files ending in abc1234.py. If the manufacturer is newly added, you should modify this sample device; otherwise, copy the four sample device files over from spacq/devices/sample/.

1. Rename all the abc1234.py files to something suitable for the device. The chosen name (excluding the .py extension) will be the module name; it should include only lowercase letters, but can also include digits and underscores starting with the second character.

   Note: Due to the constraint that the initial character of the module name may not be a digit, some model names cannot be used directly either in the module name or in the class name. For example, in the case of the Agilent 34410A, the module name is dm34410a and the class name is DM34410A where the letters “DM” are arbitrarily chosen and stand for “digital multimeter”.

2. Add the configuration details for the physical device to your test configuration (page 49).

3. Edit the mock tests (spacq/devices/<manufacturer>/mock/tests/test_mock_<model>.py) to refer to the correct tests.

4. Edit the server ² tests (spacq/devices/<manufacturer>/tests/server/test_<model>.py) to use all the functionality the interface will include.

5. Edit the device interface (spacq/devices/<manufacturer>/<model>.py) until all the server tests pass.

6. Edit the mock implementation (spacq/devices/<manufacturer>/<model>.py) until all the mock tests pass.

7. Add the new model module and mock model module to the models and mock_models lists, respectively, in the manufacturer directory __init__.py, as well as to the import lines.

   Warning: Ensure that both lists have the same length. None is an acceptable value in either list if that implementation is not available.

8.1.5 Synchronization

To allow for consistent state while performing device commands, each device contains a re-entrant lock. Every read and write operation acquires this lock; thus, multiple reads and writes are mutually excluded. In order to provide a similar mechanism for user-defined methods, the spacq.tool.box.Synchronized decorator can be used. This decorator will acquire the device lock, ensuring that other concurrently-executing threads cannot do the same, and that the atomicity of the decorated method is guaranteed for a given device instance.

² They are referred to as “server” tests because they have an external dependency (the hardware device) which acts roughly as a server to which the tests connect.
8.1.6 Graphical configuration

In the case that a device requires a graphical configuration panel, one can be added in the form of a non-modal wxPython dialog (inherited from Dialog). The dialog should reside in spacq/devices/<manufacturer>/gui/<model>.py, and its constructor must take the following arguments, in order:

1. The parent window.
2. The global store.
3. The device name as used in the global store.

The latter two values allow the dialog to find a reference to the device object itself.

In order to announce that a GUI configuration panel is available, the device class (child of AbstractDevice) must have a _gui_setup property which follows the following template:

```python
@property
def _gui_setup(self):
    try:
        from .gui.model import ModelSettingsDialog
        return ModelSettingsDialog
    except ImportError as e:
        log.debug('Could not load GUI setup for device "{0}": {1}'.format(self.name, str(e)))
    return None
```

8.2 GUI

Tools for providing applications with a graphical user interface.

8.2.1 Class hierarchy

The entire GUI framework is built on wxPython, so all widget classes ultimately inherit from a wxPython class, such as wx.Frame, wx.Panel, or wx.Dialog.

The global store provides spacq.gui.global_store.GlobalStore, a container for the spacq-related global state for an entire application. It includes separate namespaces for devices, resources, and variables, as well as a single slot for a pulse program.

The GUI toolbox provides several GUI primitives which appear frequently, such as a spacq.gui.toolbox.MessageDialog and functions for saving and loading Python-pickled and CSV files with a graphical dialog.

8.2.2 File structure

The files are laid out in the same fashion as shown in the Graphical user interface (page 19) section of the User’s Guide (page 9): they are first sorted based on the fundamental type of widget described in the file. Within that, some widgets may be classified further to restrict them to their own namespace. For example, the plots are found in spacq/gui/display/plot/ rather than just in spacq/gui/display/.

An attempt has been made to keep as much logic as possible out of the GUI code. For example, spacq.gui.action.data_capture.DataCaptureDialog only handles the progress widget, user interaction, and error messages; however, it inherits from spacq.iteration.sweep.SweepController,
which handles all the sweep logic. The former is not easily testable, but the latter is tested in
spacq.iteration.tests.test_sweep.SweepControllerTest.

### 8.2.3 Conventions

#### Dialogs

It is recommended to use `spacq.gui.tool.box.Dialog` rather than `wx.Dialog` directly, in order to
simplify disposal. An explanation for the behaviour of `wx.Dialog` exists in the `wxWidgets` documentation
(http://docs.wxwidgets.org/2.8/wx_windowdeletionoverview.html) under “Window deletion overview”.

All dialogs should be *modeless* in order to allow the user to interact with several parts of an application simultaneously.
It is often necessary to leave a dialog open and switch contexts to another part of the program; that is impossible to
achieve with modal dialogs.

#### Messaging

Due to GUI programming’s asynchronous, event-driven nature, it may be difficult to ensure that different parts of
a GUI application can communicate with each other in a timely manner. Two methods are utilized for messaging:
callbacks and a pub-sub framework.

#### Callbacks

For uni-directional, ad-hoc communication, callbacks are a simple and obvious choice. Thus, they are used extensively
in the GUI code.

For example, let us examine the common case that a dialog is launched and an action is required when the user clicks
the OK button. The dialog needs to provide the usual setup for the OK button event handling:

```python
def __init__(self, parent, ..., *args, **kwargs):
    ...
    self.Bind(wx.EVT_BUTTON, self.OnOk, ok_button)
    ...
```

However, the `OnOk` method cannot contain any logic of its own, since the dialog does not know in which context it
was called. All that needs to happen in the event handler is a call to the callback:

```python
def OnOk(self, evt=None):
    if self.ok_callback(self):
        self.Destroy()
```

where the `ok_callback` attribute is either set by `__init__` or afterwards by the caller. This structure allows the
dialog to know whether to exit (depending on the result of the callback), but does not require it to know what happens
when the button is pressed.

The event handler logic comes from whoever creates the dialog. For example:

```python
def OnAction(self, evt=None):
    var = ...

def ok_callback(dlg):
    try:
        values = dlg.GetValue()
    except ValueError as e:
        MessageDialog(self, str(e), 'Invalid value').Show()
```
Tip: As in the above example, most callbacks make use of the lexical closures that Python provides for nested functions, reducing the number of arguments that need to be passed between GUI objects.

Pub-sub

A publish-subscribe framework is used for events which must be broadcast to multiple listeners.

For example, the Data capture (page 19) panel and dialog send out data_capture.start, data_capture.data, and data_capture.stop messages to the global publisher to indicate to anybody who may be listening (there may be zero or more listeners) that certain resources are being acquired. The Measurement configuration (page 22) frames listen to whichever resource they are configured, and act accordingly when messages are received.

Subscriptions are made with a call to subscribe:

```python
pub.subscribe(self.msg_data_capture_start, 'data_capture.start')
```

Messages are sent with a call to sendMessage:

```python
pub.sendMessage('data_capture.start', name=name)
```

Warning: Since the methods associated with the subscriptions for the given topic are run in the same thread as the call to sendMessage, it is necessary to ensure thread safety (page 54) when the subscriber may perform GUI actions.

The handler must therefore have parameters which match the message being sent:

```python
def msg_data_capture_start(self, name):
    ...
```

8.2.4 Thread safety

When performing an action which affects the GUI in another thread, it is crucial to use `wx.CallAfter`. Performing the action directly, as in:

```python
self.display_label.SetValue('value')
```

will cause the GUI event loop to break non-deterministically: depending on the frequency of such calls, the app may freeze or crash within a short time, or may not do so at all. To avoid this, the above example would be rewritten as:

```python
wx.CallAfter(self.display_label.SetValue, 'value')
```
8.3 Interface

The interface package contains code for both user-interface and code-interface purposes.

8.3.1 Pulse programs

The code for dealing with pulse programs is composed of:

- a parser (`spacq.interface.pulse.parser.Parser`),
- an AST (`spacq.interface.pulse.tree.ASTNode`),
- an environment (`spacq.interface.pulse.tree.Environment`), and
- a presentation wrapper (`spacq.interface.pulse.program.Program`).

The details of pulse program handling are hidden behind the `Program` interface, which handles every step of a pulse program's life cycle from parsing to execution.

When presented with the textual representation of a pulse program, the parser does its best to transform it into an abstract syntax tree; if it cannot do so, it raises a `spacq.interface.pulse.paser.PulseSyntaxError`.

Once an AST is obtained, an empty `Environment` is created. The nodes of the AST are then visited with the `Environment` over several stages. These stages are:

1. **declarations**:
   - Populate the environment with the variable declarations.
   - Verify that the declarations make sense.

2. **values**:
   - Populate the environment with the value assignments.
   - Verify that the assigned values match the declared variables.

3. **commands**:
   - Verify that all the commands make sense given the collected declarations and values.

4. **waveforms**:
   - Based on all the collected information, generate the output waveforms.

**Warning**: The order of the stages must be preserved, since each stage makes the assumption that previous stages have been executed.

8.3.2 Resources

Resource

`spacq.interface.resources.Resource` is a generic resource with a getter and a setter, each of which can perform arbitrary code in order to get or set a value.

If a resource has units, all operations dealing with the value of the resource should use matching units. In particular, setting the value of a resource with a value whose units do not match those of the resource will raise a `TypeError`. The display units of a resource, however, have no impact on its values; the display units only affect the value displayed alongside the resource.
If a converter is supplied for a resource, the converter is used to transform user input (textual strings) into the valid type for the resource. Typical converters include `float` and `spacq.devices.tools.str_to_bool`. If a resource has units, the user input is automatically converted into a `spacq.interface.units.Quantity`; supplying a converter overrides this behaviour.

A resource may be wrapped with arbitrarily many wrappers. Wrapping and unwrapping are both non-destructive: the original resource is always unmodified, and a new Resource instance is created. For both getting and setting values, the getter and setter filters are applied in the same order they were added, excluding those which have been removed.

**Acquisition Thread**

`spacq.interface.resources.AcquisitionThread` is a threaded wrapper around a resource that allows the value of the resource to be fetched at regular intervals. This is particularly useful for live plots which show historical data.

In order to pause the acquisition, `running_lock` should be acquired from another thread (if no running lock is passed to `__init__`, pausing is disallowed); to resume, `running_lock` should be released. In order to stop the thread, `done` should be set to `True`.

### 8.3.3 Units

**SIValues**

`spacq.interface.units.SIValues` is a container for all SI prefixes (from $10^{-24}$ to $10^{24}$), all SI base units, and a selection of SI derived units.

**Quantity**

`spacq.interface.units.Quantity` is a wrapper around the Python package “quantities”. It is used internally (to communicate quantities between objects) and externally (allowing the user to enter arbitrary quantities).

**Note:** Rather than exposing the `quantities.Quantity` interface, `spacq.interface.units.Quantity` defines its own interface and uses a subset of the `quantities.Quantity` interface internally. Thus, `spacq.interface.units.Quantity` is not a drop-in substitute for `quantities.Quantity`.

### 8.3.4 Waveform generation

`spacq.interface.waveform.Generator` provides a mechanism for generating `spacq.interface.waveform.Waveform` objects. Each Generator will generate a single waveform as its methods are called; after the waveform is complete, it can be obtained via the `waveform` attribute.

**Note:** All values should be normalized to the interval [-1.0, 1.0].

### 8.4 Iteration

Tools for iterating; the core of the logic behind sweeping.
8.4.1 File structure

The package consists of only two modules: `spacq.iteration.sweep` which contains `SweepController`, and `spacq.iteration.variables` which defines input and output variables. Together, these modules can be used to provide iteration over a set of variables.

8.4.2 Sweeping

`spacq.iteration.sweep.SweepController` goes through a process of several stages, as crudely drawn in its docstring:

```
init -> next -> transition -> write -> dwell -> pulse -> read -> ramp_down -> end

^    ^    |________________________________________|
|    |    |________________________________________________________________________|
```

The ordering is approximately linear, but with some loops and skips:

- If no pulse program is defined, the `pulse` stage is skipped.
- If more items remain to be iterated over, `read` heads to `next`.
- If the sweep is continuous, `ramp_down` restarts it instead of finishing it.

Those steps which deal with accessing resources (`transition`, `write`, `read`, `ramp_down`) do so in parallel, using as many concurrent `threading.Thread` objects as necessary.

The sweeping process can be interrupted at any time for many reasons; some of these include: user error, device error, and the user pressing the “Cancel” button. In the case that it is interrupted, the sweep simply proceeds to either the `ramp_down` or the `end` stage, depending on whether the interruption is fatal. In the case of a fatal interruption, the `ramp_down` stage cannot be expected to succeed (for example, if writing to a resource failed), so it is skipped.
This section describes the execution of pulse programs on the AWG and oscilloscope.

See also:

*Pulse programs* (page 55)

### 9.1 Device initialization

When a sweep begins, the method `spacq.iteration.sweep.SweepController.init()` is run, which performs several steps, one of which is device initialization. At this point, if a pulse program has been configured, the AWG is disabled and placed into “triggered” mode, and its sampling rate is set to that of the pulse program.

### 9.2 Device utilization

If a pulse program is configured, the method `spacq.iteration.sweep.SweepController.pulse()` is called by `dwell()`. If no output channels have been configured for the AWG (ie. no waveform outputs have been mapped to AWG channels), this stage does nothing. Otherwise, the following sequence of events occurs:

1. The waveforms are generated using the latest values written to the pulse program resources.
2. The AWG is configured:
   a. It is disabled, and its channels are cleared of waveforms.
   b. Each output channel has its waveform loaded.
   c. All used channels are enabled.
   d. The AWG itself is enabled.
3. The oscilloscope is configured:
   a. It is disabled.
   b. If averaging has been requested, it is placed into “FastFrame” mode (page 50) with an average summary frame. Otherwise, “FastFrame” mode is disabled.
   c. It is put into single sequence mode.
4. Execution pauses until the AWG and oscilloscope have both completed carrying out the above steps.
5. The oscilloscope is enabled and a delay of 1 s occurs to allow the oscilloscope trigger to ready.

**Note:** The oscilloscope cannot be enabled as part of the above configuration process because the setup procedure of the AWG may trigger the acquisition of an invalid waveform.

6. The trigger loop is executed the specified number of times (“times to average”):
   (a) The AWG is triggered.
   (b) Execution pauses until the trigger event has executed.
   (c) The “post-trigger delay” is waited.

7. Execution pauses until the oscilloscope has finished capturing. This includes performing the averaging if in “FastFrame” mode.

8. The number of acquired frames is verified.