naqslab_devices

Release 0.2.7

dihm

Mar 04, 2020

CONTENTS:

This library is a collection of third-party devices for the labscript_suite experimental control system.

CHAPTER

ONE

INSTALLATION

Prerequiste: labscript_devices >= 2.2.0

Clone this repository into the labscript_suite directory. Typically into `userlib` or the main directory level.

Next, modify your labconfig.ini file to recognize the module by adding the following entry to the `[DEFAULT]` block:

```
user_devices = naqslab_devices
```

If more than one module of 3rd party devices are used, put all module names as a comma separated list.

CHAPTER

TWO

USAGE

Invoke in labscript scripts just like other labscript devices:

```
from naqslab_devices import ScopeChannel
from naqslab_devices.KeysightXSeries.labscript_device import KeysightXScope
```

Details for how to use each device are contained in the *detailed documentation* listings.

CHAPTER

THREE

DEVICES

Directory of all device classes in this repository.

The labscript primitive subclasses are derivatives of the labscript-provided children classes used by devices in this repository.

There are two parent classes that are not directly used, but rather provide templates for creating new devices. First is the *VISA* class that templates communication with devices through the VISA communication protocol. This uses the PyVISA python wrapper. The second is the *SignalGenerator* class that uses the *VISA* class to template CW frequency generators.

There are two thin subclasses of the labscript_devices.PulseBlaster_No_DDS class: *PulseBlasterESRPro300* and *PulseBlaster_No_DDS_200*. They exist simply to enforce the correct core clock frequency and clock limits without any other change in functionality from the parent.

Other device classes control particular series of devices and implement functional control of their hardward to varying degrees. In general, the design philosophy is that if the device class does not set an option, it will not be interfered with when using the device class to control the instrument. This means that custom settings and configurations of each device can be used by setting them manually at the device front panel without the device class interfering.

3.1 `labscript` Primitive Subclasses

3.1.1 Overview

These are subclasses of labscript primitive objects for specific use with the devices in this module.

`StaticFreqAmp` is used by the `NovaTechDDS` devices. `ScopeChannel` is used by the KeysightXSeries and TektronixTDS oscilloscope classes. `CounterScopeChannel` is used exclusively by the KeysightXSeries oscilloscopes.

`StaticFreqAmp`	This instatiates a static frequency output channel.
`ScopeChannel`	This instantiates a scope channel to acquire during a buffered shot.
`CounterScopeChannel`	This instantiates a counter scope channel to acquire during a buffered shot.

3.1.2 Detailed Documentation

class naqslab_devices.**ScopeChannel**(*name*, *parent_device*, *connection*)

Bases: `labscript.labscript.AnalogIn`

This instantiates a scope channel to acquire during a buffered shot.

Parameters

- **name** (*str*) – Name to assign channel

- **parent_device** (*obj*) – Handle to parent device
- **connection** (*str*) – Which physical scope channel is acquiring. Generally of the form 'Channel n' where n is the channel label.

description = 'Scope Acquisition Channel Class'

acquire()
Inform BLACS to save data from this channel.

Note that the parent_device controls when the acquisition trigger is sent.

add_device(*device*)

allowed_children = None

generate_code(*hdf5_file*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.**CounterScopeChannel**(*name*, *parent_device*, *connection*)
Bases: *naqslab_devices.ScopeChannel*

This instantiates a counter scope channel to acquire during a buffered shot.

Parameters

- **name** (*str*) – Name to assign channel
- **parent_device** (*obj*) – Handle to parent device
- **connection** (*str*) – Which physical scope channel is acquiring. Generally of the form 'Channel n' where n is the channel label.

description = 'Scope Acquisition Channel Class with Pulse Counting'

count(*typ*, *pol*)
Register a pulse counter operation for this channel.

Parameters

- **typ** (*str*) – count 'pulse' or 'edge'
- **pol** (*str*) – reference to 'pos' or 'neg' edges

acquire()
Inform BLACS to save data from this channel.

Note that the parent_device controls when the acquisition trigger is sent.

add_device(*device*)

allowed_children = None

generate_code(*hdf5_file*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.**StaticFreqAmp**(*name*, *parent_device*, *connection*, *freq_limits=None*, *freq_conv_class=None*, *freq_conv_params={}*, *amp_limits=None*, *amp_conv_class=None*, *amp_conv_params={}*)
Bases: labscript.labscript.StaticDDS

This instatiates a static frequency output channel.

Frequency and amplitude limits set here will supercede those dictated by the device class, but only when compiling a shot with runmanager. Static update limits are enforced by the BLACS Tab for the parent device.

Parameters

- **name** (*str*) – Name to assign output channel
- **parent_device** (*obj*) – Handle to parent device
- **connection** (*str*) – Which physical channel to use on parent device. Typically only 'Channel 0' is available.
- **freq_limits** (*tuple*) – Set (min,max) output frequencies in BLACS front panel units.
- **freq_conv_class** (*obj*) – Custom conversion class to use
- **freq_conv_params** (*dict*) – Parameters to conversion class
- **amp_limits** (*tuple*) – Set (min,max) output amplitude in BLACS front panel units.
- **amp_conv_class** (*obj*) – Custom convsersion class to use

- **amp_conv_params** (*dict*) – Parameters to conversion class

description = 'Frequency Source class for Signal Generators'

allowed_children = [<class 'labscript.labscript.StaticAnalogQuantity'>]

setphase(*value*, *units=None*)
Overridden from StaticDDS so as not to provide phase control, which is generally not supported by `SignalGenerator` devices.

enable()
overridden from StaticDDS so as not to provide time resolution - output can be enabled or disabled only at the start of the shot

disable()
overridden from StaticDDS so as not to provide time resolution - output can be enabled or disabled only at the start of the shot

add_device(*device*)

generate_code(*hdf5_file*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

setamp(*value*, *units=None*)

setfreq(*value*, *units=None*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

3.2 VISA

3.2.1 Overview

This parent class establishes the standard communication protocol for all devices that use the VISA communication library. It relies on the PyVISA python wrapper.

naqslab_devices.VISA.blacs_tab	Boiler plate blacs_tab for VISA instruments.
naqslab_devices.VISA.blacs_worker	Boiler plate BLACS_worker for VISA instruments.

Continued on next page

Table 2 – continued from previous page

naqslab_devices.VISA.labscript_device	Boiler plate labscript_device for VISA instruments.
naqslab_devices.VISA.register_classes	Sets which BLACS_tab belongs to each labscript device.

3.2.2 Detailed Documentation of naqslab_devices.VISA

VISA.blacs_tab

Boiler plate blacs_tab for VISA instruments.

Defines the common STBstatus.ui widget all devices use to report their current status.

class naqslab_devices.VISA.blacs_tab.**VISATab**(**args*, ***kwargs*)

Bases: blacs.device_base_class.DeviceTab

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

status_byte_labels = {'bit 0': 'bit 0 label', 'bit 1': 'bit 1 label', 'bit 2': 'b

status_widget = 'STBstatus.ui'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

initialise_GUI()

Loads the standard STBstatus.ui widget and sets the worker defined in __init__

status_monitor(**args*, ***kwargs*)

send_clear(**args*, ***kwargs*)

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)

Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)

Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

VISA.blacs_worker

Boiler plate BLACS_worker for VISA instruments.

Inheritors use the same communication protocol, but override the command syntax.

class naqslab_devices.VISA.blacs_worker.**VISAWorker**(**args*, ***kwargs*)
Bases: blacs.tab_base_classes.Worker

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

check_remote_values()

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from read_stb

Returns Status byte dictionary as formatted in VISATab

Return type dict

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in VISATab

Return type dict

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns *VISAWorker.check_remote_values()*

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

abort_buffered()
Special abort shot code belongs here.

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

shutdown()
Closes VISA connection to device.

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

VISA.labscript_device

Boiler plate labscript_device for VISA instruments.

class naqslab_devices.VISA.labscript_device.**VISA**(*name*, *parent_device*, *VISA_name*, ***kwargs*)

Bases: labscript.labscript.Device

Base VISA labscript_device class.

Inheritors should call VISA.__init__() in their own __init__() method. Generate_code must be overridden.

Parameters

- **name** (*str*) – name of device in connectiontable

- **parent_device** (*obj*) – Handle to any parent device.
- **VISA_name** (*str*) – Can be full VISA connection string or NI-MAX alias.

description = 'VISA Compatible Instrument'

allowed_children = []

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

add_device(*device*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

VISA.register_classes

Sets which BLACS_tab belongs to each labscript device.

3.3 SignalGenerator

3.3.1 Overview

This is a parent class for static RF Signal Generators with frequency and amplitude control and use the IEEE 488 or SCPI command protocols. Device communication is handled using the VISA communication protocol.

naqslab_devices.SignalGenerator.Models
naqslab_devices.SignalGenerator.blacs_tab
naqslab_devices.SignalGenerator.blacs_worker

Continued on next page

Table 3 – continued from previous page

naqslab_devices.SignalGenerator.labscript_device	
naqslab_devices.SignalGenerator.register_classes	Configures which BLACS_tab goes to which labscript_device.

3.3.2 Models

Currently covered models include:

HP_8642A
HP_8643A
HP_8648
RS_SMA100B
RS_SMF100A
RS_SMHU
KeysightSigGens

3.3.3 Adding a Signal Generator

In order to add a new model of signal generator one must:

1. Add a subclass `SignalGenerator.labscript_device` in `SignalGenerator.Models` which provides the operational limits.
2. Subclass `SignalGenerator.blacs_tab` and `SignalGenerator.blacs_worker` with the operational details for communicating with the device (namely command syntax and status byte definitions).
3. Add an entry in `SignalGenerator.register_classes` that links the labscript_device to the correct blacs_tab. Note that multiple labscript_devices can use the same blacs_tab/blacs_worker.

3.3.4 Detailed Documentation

class naqslab_devices.SignalGenerator.BLACS.HP_8642A.**HP_8642ATab**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_tab.SignalGeneratorTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

```
base_units = {'amp':  'dBm', 'freq':  'MHz'}

base_min = {'amp':  -140, 'freq':  0.1}

base_max = {'amp':  20, 'freq':  1057.5}

base_step = {'amp':  0.1, 'freq':  1}

base_decimals = {'amp':  1, 'freq':  6}

status_byte_labels = {'bit 0':  'End of Sweep', 'bit 1':  'Hardware Error', 'bit 2'

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'
```

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.HP_8642A.**HP_8642AWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_worker.SignalGeneratorWorker*

scale_factor = 1000000.0

amp_scale_factor = 1.0

freq_write_string = 'FR {:.0f} HZ'

freq_query_string = 'FROA'

freq_parser(*freq_string*)
Frequency Query string parser for HP8642A freq_string format is FR sddddddddd.0 HZ Returns float in instrument units, Hz (i.e. needs scaling to base_units)

amp_write_string = 'AP {:.1f} DM'

amp_query_string = 'APOA'

amp_parser(*amp_string*)
Amplitude Query string parser for HP8642A amp_string format is AP sddd.d DM Returns float in instrument units, dBm

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in VISATab

Return type dict

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from read_stb

Returns Status byte dictionary as formatted in VISATab

Return type dict

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns VISAWorker.check_remote_values()

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.BLACS.HP_8643A.**HP_8643ATab**(**args*, ***kwargs*)
Bases: *naqslab_devices.SignalGenerator.blacs_tab.SignalGeneratorTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_units = {'amp': 'dBm', 'freq': 'MHz'}

base_min = {'amp': -137, 'freq': 0.26}

base_max = {'amp': 13, 'freq': 1030}

base_step = {'amp': 0.1, 'freq': 1}

base_decimals = {'amp': 2, 'freq': 8}

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'RQC', 'bit 2': 'Qu

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.HP_8643A.**HP_8643AWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_worker.SignalGeneratorWorker*

scale_factor = 1000000.0

amp_scale_factor = 1.0

init()
Calls parent init and sends device specific initialization commands

freq_write_string = 'FREQ:CW {:.2f} HZ'

freq_query_string = 'FREQ:CW?'

freq_parser(*freq_string*)
Frequency Query string parser for HP8643A freq_string format is float, in Hz Returns float in instrument units, Hz (i.e. needs scaling to base_units)

amp_write_string = 'AMPL:LEV {:.2f} DBM'

amp_query_string = 'AMPL:LEV?'

amp_parser(*amp_string*)
Amplitude Query string parser for HP8643A amp_string format is float in configured units (dBm by default) Returns float in instrument units, dBm

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.BLACS.HP_8648.**HP_8648ATab**(**args*, ***kwargs*)
Bases: *naqslab_devices.SignalGenerator.blacs_tab.SignalGeneratorTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_units = {'amp': 'dBm', 'freq': 'MHz'}

base_min = {'amp': -136, 'freq': 0.1}

base_max = {'amp': 20, 'freq': 1000}

base_step = {'amp': 1, 'freq': 1}

base_decimals = {'amp': 1, 'freq': 9}

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'RQC', 'bit 2': 'Qu

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.HP_8648.**HP_8648BTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.BLACS.HP_8648.HP_8648ATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_min = {'amp': -136, 'freq': 0.009}

base_max = {'amp': 20, 'freq': 2000}

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

base_decimals = {'amp': 1, 'freq': 9}

base_step = {'amp': 1, 'freq': 1}

base_units = {'amp': 'dBm', 'freq': 'MHz'}

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'RQC', 'bit 2': 'Qu

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.HP_8648.**HP_8648CTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.BLACS.HP_8648.HP_8648ATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_min = {'amp': -136, 'freq': 0.009}

base_max = {'amp': 20, 'freq': 3200}

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

base_decimals = {'amp': 1, 'freq': 9}

base_step = {'amp': 1, 'freq': 1}

base_units = {'amp': 'dBm', 'freq': 'MHz'}

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'RQC', 'bit 2': 'Qu

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.HP_8648.**HP_8648DTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.BLACS.HP_8648.HP_8648ATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_min = {'amp': -136, 'freq': 0.009}

base_max = {'amp': 20, 'freq': 4000}

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

base_decimals = {'amp': 1, 'freq': 9}

base_step = {'amp': 1, 'freq': 1}

base_units = {'amp': 'dBm', 'freq': 'MHz'}

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'RQC', 'bit 2': 'Qu

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.HP_8648.**HP_8648Worker**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_worker.SignalGeneratorWorker*

scale_factor = 1000000.0

amp_scale_factor = 1.0

init()
Calls parent init and sends device specific initialization commands

freq_write_string = 'FREQ:CW {:.3f} HZ'

freq_query_string = 'FREQ:CW?'

freq_parser(*freq_string*)
Frequency Query string parser for HP8648 freq_string format is float, in Hz Returns float in instrument units, Hz (i.e. needs scaling to base_units)

amp_write_string = 'POW:AMPL {:.1f} DBM'

amp_query_string = 'POW:AMPL?'

amp_parser(*amp_string*)
Amplitude Query string parser for HP8648 amp_string format is float in configured units (dBm by default) Returns float in instrument units, dBm

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.BLACS.RS_SMA100B.**RS_SMA100BTab**(**args*, ***kwargs*)
Bases: *naqslab_devices.SignalGenerator.blacs_tab.SignalGeneratorTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_units = {'amp': 'dBm', 'freq': 'GHz'}

base_min = {'amp': -145, 'freq': 8e-06}

base_max = {'amp': 35, 'freq': 20}

base_step = {'amp': 1, 'freq': 0.1}

base_decimals = {'amp': 2, 'freq': 9}

status_byte_labels = {'bit 0': 'Unused', 'bit 1': 'Unused', 'bit 2': 'Error Queue

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.RS_SMA100B.**RS_SMA100BWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_worker.SignalGeneratorWorker*

scale_factor = 1000000000.0

amp_scale_factor = 1.0

freq_write_string = 'FREQ:CW {:.0f}HZ'

freq_query_string = 'FREQ:CW?'

freq_parser(*freq_string*)
Frequency Query string parser for SMF100A freq_string format is ddddddddddd Returns float in instrument units, Hz (i.e. needs scaling to base_units)

amp_write_string = 'POW:POW {:.2f}dBm'

amp_query_string = 'POW?'

amp_parser(*amp_string*)
Amplitude Query string parser for SMF100A Returns float in instrument units, dBm

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable

- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.BLACS.RS_SMF100A.**RS_SMF100ATab**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_tab.SignalGeneratorTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_units = {'amp': 'dBm', 'freq': 'GHz'}

base_min = {'amp': -26, 'freq': 1e-06}

base_max = {'amp': 18, 'freq': 22}

base_step = {'amp': 1, 'freq': 0.1}

base_decimals = {'amp': 1, 'freq': 6}

status_byte_labels = {'bit 0': 'Unused', 'bit 1': 'Unused', 'bit 2': 'Error Queue

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.RS_SMF100A.**RS_SMF100AWorker**(**args*, ***kwargs*)
Bases: *naqslab_devices.SignalGenerator.blacs_worker.SignalGeneratorWorker*

scale_factor = 1000000000.0

amp_scale_factor = 1.0

freq_write_string = 'FREQ:CW {:.0f}HZ'

freq_query_string = 'FREQ?'

freq_parser(*freq_string*)
Frequency Query string parser for SMF100A freq_string format is ddddddddddd Returns float in instrument units, Hz (i.e. needs scaling to base_units)

amp_write_string = 'POW:POW {:.2f}dBm'

amp_query_string = 'POW?'

amp_parser(*amp_string*)
Amplitude Query string parser for SMF100A Returns float in instrument units, dBm

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in VISATab

Return type dict

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.BLACS.RS_SMHU.**RS_SMHUTab**(**args*, ***kwargs*)
Bases: *naqslab_devices.SignalGenerator.blacs_tab.SignalGeneratorTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_units = {'amp': 'dBm', 'freq': 'MHz'}

base_min = {'amp': -140, 'freq': 0.1}

base_max = {'amp': 13, 'freq': 4320}

base_step = {'amp': 0.1, 'freq': 1}

base_decimals = {'amp': 1, 'freq': 7}

status_byte_labels = {'bit 0': 'OPC', 'bit 1': 'SRQ', 'bit 2': 'Query Error', 'bi

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.RS_SMHU.**RS_SMHUWorker**(**args*, ***kwargs*)
Bases: *naqslab_devices.SignalGenerator.blacs_worker.SignalGeneratorWorker*

scale_factor = 1000000.0

amp_scale_factor = 1.0

init()
Calls parent init and sends device specific initialization commands

freq_write_string = 'RF {:.1f}HZ'

freq_query_string = 'RF?'

freq_parser(*freq_string*)
Frequency Query string parser for RS SMHU freq_string format is sddddddddddd.d Returns float in instrument units, Hz (i.e. needs scaling to base_units)

amp_write_string = 'LEVEL:RF {:.1f}DBM'

amp_query_string = 'LEVEL:RF?'

amp_parser(*amp_string*)
Amplitude Query string parser for RS SMHU amp_string format is sddd.d Returns float in instrument units, dBm

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in VISATab

Return type dict

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from read_stb

Returns Status byte dictionary as formatted in VISATab

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns VISAWorker.check_remote_values()

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.BLACS.KeysightSigGens.**KeysightSigGenTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_tab.SignalGeneratorTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'RQC', 'bit 2': 'Qu

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

base_decimals = {'amp': 1, 'freq': 6}

base_max = {'amp': 20, 'freq': 1057.5}

base_min = {'amp': -140, 'freq': 0.1}

base_step = {'amp': 0.1, 'freq': 1}

base_units = {'amp': 'dBm', 'freq': 'MHz'}

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.KeysightSigGens.**E8257NTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.BLACS.KeysightSigGens.KeysightSigGenTab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_units = {'amp': 'dBm', 'freq': 'GHz'}

base_min = {'amp': -105, 'freq': 0.01}

base_max = {'amp': 20, 'freq': 40.0}

base_step = {'amp': 1, 'freq': 1}

base_decimals = {'amp': 1, 'freq': 12}

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)

Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)

Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'RQC', 'bit 2': 'Qu

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.BLACS.KeysightSigGens.**KeysightSigGenWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.SignalGenerator.blacs_worker.SignalGeneratorWorker*

Generic BLACS worker for Modern Keysight/Agilent/HP CW Signal Generators

This class defines the common frequency/amplitude read/write commands as well as general device configuration and communication. It uses the ESR byte instead of the standard STB byte for intstrument communications.

The *scale_factor* and *amp_scale_factor* are taken from the labscript device specification.

init()
Calls parent init and sends device specific initialization commands

freq_write_string = 'FREQ:CW {:.3f} HZ'

freq_query_string = 'FREQ:CW?'

freq_parser(*freq_string*)
Frequency Query string parser for Keysight Sig Gens freq_string format is float, in Hz Returns float in instrument units, Hz (i.e. needs scaling to base_units)

amp_write_string = 'POW:AMPL {:.1f} DBM'

amp_query_string = 'POW:AMPL?'

amp_parser(*amp_string*)
Amplitude Query string parser for Keysight Sig Gens amp_string format is float in configured units (dBm by default) Returns float in instrument units, dBm

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

amp_scale_factor = 1.0

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

scale_factor = 1000000.0

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.Models.**RS_SMF100A**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'Rhode & Schwarz SMF100A Signal Generator'

scale_factor = 1000000000.0

freq_limits = (100000.0, 22000000000.0)

amp_scale_factor = 1.0

amp_limits = (-26, 18)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**RS_SMA100B**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'Rhode & Schwarz SMA100B Signal Generator'

scale_factor = 1000000000.0

freq_limits = (8000.0, 20000000000.0)

amp_scale_factor = 1.0

amp_limits = (-145, 35)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**RS_SMHU**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'RS SMHU Signal Generator'

scale_factor = 1000000.0

freq_limits = (100000.0, 4320000000.0)

amp_scale_factor = 1.0

amp_limits = (-140, 13)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**HP_8643A**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'HP 8643A Signal Generator'

scale_factor = 1000000.0

freq_limits = (260000.0, 1030000000.0)

amp_scale_factor = 1.0

amp_limits = (-137, 13)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**HP_8642A**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'HP 8642A Signal Generator'

scale_factor = 1000000.0

freq_limits = (100000.0, 1057500000.0)

amp_scale_factor = 1.0

amp_limits = (-140, 20)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**HP_8648A**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'HP 8648A Signal Generator'

scale_factor = 1000000.0

freq_limits = (100000.0, 1000000000.0)

amp_scale_factor = 1.0

amp_limits = (-136, 20)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**HP_8648B**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'HP 8648B Signal Generator'

scale_factor = 1000000.0

freq_limits = (9000.0, 2000000000.0)

amp_scale_factor = 1.0

amp_limits = (-136, 20)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**HP_8648C**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'HP 8648C Signal Generator'

scale_factor = 1000000.0

freq_limits = (9000.0, 3200000000.0)

amp_scale_factor = 1.0

amp_limits = (-136, 20)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**HP_8648D**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'HP 8648D Signal Generator'

scale_factor = 1000000.0

freq_limits = (9000.0, 4000000000.0)

amp_scale_factor = 1.0

amp_limits = (-136, 20)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.Models.**E8257N**(*name*, *VISA_name*)
Bases: *naqslab_devices.SignalGenerator.labscript_device.SignalGenerator*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'Keysight E8257N Signal Generator'

scale_factor = 1000000000.0

freq_limits = (10000000.0, 40000000000.0)

amp_scale_factor = 1.0

amp_limits = (-105, 20)

add_device(*device*)

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.SignalGenerator.blacs_tab.**SignalGeneratorTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_tab.VISATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

base_units = {'amp': 'dBm', 'freq': 'MHz'}

base_min = {'amp': -140, 'freq': 0.1}

base_max = {'amp': 20, 'freq': 1057.5}

base_step = {'amp': 0.1, 'freq': 1}

base_decimals = {'amp': 1, 'freq': 6}

status_byte_labels = {'bit 0': 'bit 0 label', 'bit 1': 'bit 1 label', 'bit 2': 'b

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.SignalGenerator.blacs_worker.**SignalGeneratorWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_worker.VISAWorker*

scale_factor = 1000000.0

amp_scale_factor = 1.0

freq_write_string = ''

freq_query_string = ''

freq_parser(*freq_string*)
Frequency Query string parser Needs to be over-ridden

amp_write_string = ''

amp_query_string = ''

amp_parser(*amp_string*)
Amplitude Query string parser Needs to be over-ridden

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

check_remote_values()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_status()
Reads the Status Byte Register of the VISA device.

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

class naqslab_devices.SignalGenerator.labscript_device.**SignalGenerator**(*name*, *VISA_name*)

Bases: *naqslab_devices.VISA.labscript_device.VISA*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'Signal Generator'

allowed_children = [<class 'naqslab_devices.StaticFreqAmp'>]

scale_factor = 1000000.0

freq_limits = ()

amp_scale_factor = 1.0

amp_limits = ()

quantise_freq(*data*, *device*)
Quantize the frequency in units of Hz and check it's within bounds

quantise_amp(*data*, *device*)
Quantize the amplitude in units of dBm and check it's within bounds

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

add_device(*device*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

Configures which BLACS_tab goes to which labscript_device.

3.4 PulseBlasterESRPro300

3.4.1 Overview

This is a thin subclass of the PulseBlaster_No_DDS labscript_device. It merely configures the correct clock speed, digital output number, and clock resolution limits.

naqslab_devices. *PulseBlasterESRPro300.blacs_tab*
naqslab_devices. *PulseBlasterESRPro300.blacs_worker*
naqslab_devices. *PulseBlasterESRPro300.* *labscript_device*
naqslab_devices. *PulseBlasterESRPro300.* *register_classes*
naqslab_devices. *PulseBlasterESRPro300.* *runviewer_parser*

3.4.2 Detailed Documentation of naqslab_devices.PulseBlasterESRPro300

PulseBlasterESRPro300.blacs_tab

class naqslab_devices.PulseBlasterESRPro300.blacs_tab.**PulseBlasterESRPro300Tab**(**args*, ***kwargs*)

Bases: labscript_devices.PulseBlaster_No_DDS.Pulseblaster_No_DDS_Tab

num_DO = 24

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()

initialise_workers()

labscript_device_class_name = 'PulseBlaster_No_DDS'

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

reset(**args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

stop(**args*, ***kwargs*)

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

PulseBlasterESRPro300.blacs_worker

class naqslab_devices.PulseBlasterESRPro300.blacs_worker.**PulseBlasterESRPro300Worker**(**ar*
***k*

Bases: labscript_devices.PulseBlaster_No_DDS.PulseblasterNoDDSWorker

core_clock_freq = 300.0

ESRPro = True

abort_buffered()

abort_transition_to_buffered()

check_status()

init()

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*values*)

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

start_run()

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)

transition_to_manual()

PulseBlasterESRPro300.labscript_device

class naqslab_devices.PulseBlasterESRPro300.labscript_device.**PulseBlasterESRPro300**(*name*,
trig-
ger_de
trig-
ger_co
*board_*
firmwa
pro-
gram-
ming_s
*pulse_*
max_in
time_b
time_b
***kwa*

Bases: labscript_devices.PulseBlaster_No_DDS.PulseBlaster_No_DDS

description = 'SpinCore PulseBlaster ESR-PRO-300'

clock_limit = 30000000.0

clock_resolution = 4e-09

n_flags = 24

add_device(*device*)

allowed_children = [<class 'labscript.labscript.Pseudoclock'>]

convert_to_pb_inst(*dig_outputs*, *dds_outputs*, *freqs*, *amps*, *phases*)

core_clock_freq = 100

property direct_outputs

do_checks(*outputs*)
Basic error checking to ensure the user's instructions make sense

flag_is_clock(*flag*)

flag_valid(*flag*)

generate_code(*hdf5_file*)

generate_registers(*hdf5_file*, *dds_outputs*)

get_all_children()

get_all_outputs()

get_direct_outputs()
Finds out which outputs are directly attached to the PulseBlaster

get_flag_number(*connection*)

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property is_master_pseudoclock

minimum_recovery_time = 0

offset_instructions_from_trigger(*outputs*)

property parent_clock_line

pb_instructions = {'BRANCH': 6, 'CONTINUE': 0, 'END_LOOP': 3, 'LONG_DELAY': 7, 'LOOP

property pseudoclock

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_initial_trigger_time(*t*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

trigger(*t*, *duration*, *wait_delay=0*)
Ask the trigger device to produce a digital pulse of a given duration to trigger this pseudoclock

trigger_delay = 2.5e-07

trigger_edge_type = 'falling'

trigger_minimum_duration = 0

wait_delay = 1e-07

write_pb_inst_to_h5(*pb_inst*, *hdf5_file*)

PulseBlasterESRPro300.register_classes

PulseBlasterESRPro300.runviewer_parser

class naqslab_devices.PulseBlasterESRPro300.runviewer_parser.**PulseBlasterESRPro300Parser**

Bases: labscript_devices.PulseBlaster_No_DDS.PulseBlaster_No_DDS_Parser

num_flags = 24

get_traces(*add_trace*, *parent=None*)

labscript_device_class_name = 'PulseBlaster_No_DDS'

num_dds = 0

3.5 PulseBlaster_No_DDS_200

3.5.1 Overview

This is a thin subclass of the PulseBlaster_No_DDS labscript_device. It merely configures the correct clock speed and clock resolution limits for our custom 200 MHz clocked USB PulseBlaster board.

naqslab_devices.PulseBlaster_No_DDS_200.blacs_tab
naqslab_devices.PulseBlaster_No_DDS_200.blacs_worker
naqslab_devices.PulseBlaster_No_DDS_200.labscript_device
naqslab_devices.PulseBlaster_No_DDS_200.register_classes
naqslab_devices.PulseBlaster_No_DDS_200.runviewer_parser

3.5.2 Detailed Documentation of naqslab_devices.PulseBlaster_No_DDS_200

PulseBlaster_No_DDS_200.blacs_tab

class naqslab_devices.PulseBlaster_No_DDS_200.blacs_tab.**PulseBlaster_No_DDS_200_Tab**(**args*, ***kwargs*)

Bases: labscript_devices.PulseBlaster_No_DDS.Pulseblaster_No_DDS_Tab

num_DO = 24

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()

initialise_workers()

labscript_device_class_name = 'PulseBlaster_No_DDS'

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

reset(**args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

stop(**args*, ***kwargs*)

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

PulseBlaster_No_DDS_200.blacs_worker

class naqslab_devices.PulseBlaster_No_DDS_200.blacs_worker.**PulseblasterNoDDS200Worker**(**a*
****

Bases: labscript_devices.PulseBlaster_No_DDS.PulseblasterNoDDSWorker

core_clock_freq = 200.0

abort_buffered()

abort_transition_to_buffered()

check_status()

init()

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*values*)

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

start_run()

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)

transition_to_manual()

PulseBlaster_No_DDS_200.labscript_device

class naqslab_devices.PulseBlaster_No_DDS_200.labscript_device.**PulseBlaster_No_DDS_200**(*n*
t
g
t
g
b
f
p
g
n
p
n
t
t
*

Bases: labscript_devices.PulseBlaster_No_DDS.PulseBlaster_No_DDS

A thin subclass of labscript_devices.PulseBlaster_No_DDS.

It's only purpose is to set the core clock frequency to 200 MHz for our one custom USB pulseblaster device.

description = 'SpinCore PulseBlaster USB with 200 MHz clock'

clock_limit = 17200000.0

clock_resolution = 1e-08

core_clock_freq = 200

n_flags = 24

add_device(*device*)

allowed_children = [<class 'labscript.labscript.Pseudoclock'>]

convert_to_pb_inst(*dig_outputs*, *dds_outputs*, *freqs*, *amps*, *phases*)

property direct_outputs

do_checks(*outputs*)
Basic error checking to ensure the user's instructions make sense

flag_is_clock(*flag*)

flag_valid(*flag*)

generate_code(*hdf5_file*)

generate_registers(*hdf5_file*, *dds_outputs*)

get_all_children()

get_all_outputs()

get_direct_outputs()
Finds out which outputs are directly attached to the PulseBlaster

get_flag_number(*connection*)

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property is_master_pseudoclock

minimum_recovery_time = 0

offset_instructions_from_trigger(*outputs*)

property parent_clock_line

pb_instructions = {'BRANCH': 6, 'CONTINUE': 0, 'END_LOOP': 3, 'LONG_DELAY': 7, 'LOOP

property pseudoclock

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_initial_trigger_time(*t*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

trigger(*t*, *duration*, *wait_delay=0*)
Ask the trigger device to produce a digital pulse of a given duration to trigger this pseudoclock

trigger_delay = 2.5e-07

trigger_edge_type = 'falling'

trigger_minimum_duration = 0

wait_delay = 1e-07

write_pb_inst_to_h5(*pb_inst*, *hdf5_file*)

PulseBlaster_No_DDS_200.register_classes

PulseBlaster_No_DDS_200.runviewer_parser

class naqslab_devices.PulseBlaster_No_DDS_200.runviewer_parser.**PulseBlaster_No_DDS_200_P**

Bases: labscript_devices.PulseBlaster_No_DDS.PulseBlaster_No_DDS_Parser

num_flags = 24

get_traces(*add_trace*, *parent=None*)

labscript_device_class_name = 'PulseBlaster_No_DDS'

num_dds = 0

3.6 KeysightXSeries

3.6.1 Overview

This devices covers Keysight X-series oscilloscopes. It has been explicitely tested with 1000 and 3000 series 'scopes. Other series should be simple to implement.

naqslab_devices.KeysightXSeries.blacs_tab
naqslab_devices.KeysightXSeries.blacs_worker
naqslab_devices.KeysightXSeries.labscript_device
naqslab_devices.KeysightXSeries.register_classes

3.6.2 Detailed Documentation of naqslab_devices.KeysightXSeries

KeysightXSeries.blacs_tab

class naqslab_devices.KeysightXSeries.blacs_tab.**KeysightXScopeTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_tab.VISATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

status_byte_labels = {'bit 0': 'Operation Complete', 'bit 1': 'Unused', 'bit 2':

initialise_GUI()

Loads the standard STBstatus.ui widget and sets the worker defined in __init__

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)

Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)

Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

KeysightXSeries.blacs_worker

class naqslab_devices.KeysightXSeries.blacs_worker.**KeysightXScopeWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_worker.VISAWorker*

setup_string = '*ESE 60;*SRE 32;*CLS;:WAV:BYT MSBF;UNS ON;POIN:MODE RAW'

esr_mask = 60

read_analog_parameters_string = ':WAV:FORM WORD;SOUR CHAN{0:d};PRE?'

read_dig_parameters_string = ':WAV:FORM BYTE;SOUR POD{0:d};PRE?'

read_waveform_string = ':WAV:DATA?'

read_counter_string = ':MEAS:{0:s}{1:s}? CHAN{2:d}'

model_ident = 'SO-X'

dig_command = ':DIG'

analog_waveform_parser(*raw_waveform_array*, *y0*, *dy*, *yoffset*)
Parses the numpy array from the analog waveform query.

digital_pod_parser(*raw_pod_array*)
Unpacks the bits for a pod array Columns returned are in bit order [7,6,5,4,3,2,1,0]

error_parser(*error_return_string*)
Parses the strings returned by :SYST:ERR? Returns int_code, err_string

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
This configures counters, if any are defined, as well as optional compression options for saved data traces.

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

check_status()
Periodically called by BLACS to check to status of the scope.

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

KeysightXSeries.labscript_device

class naqslab_devices.KeysightXSeries.labscript_device.**KeysightXScope**(*name*, *VISA_name*, *trigger_device*, *trigger_connection*, *num_AI=4*, *DI=True*, *trigger_duration=0.001*, *compression=None*, *compression_opts=None*, *shuffle=False*, ***kwargs*)

Bases: labscript.labscript.TriggerableDevice

VISA_name can be full VISA connection string or NI-MAX alias. Trigger Device should be fast clocked device. num_AI sets number of analog input channels, default 4 DI sets if DI are present, default True trigger_duration set scope trigger duration, default 1ms Compression of traces in h5 file controlled by: compression: 'lzf', 'gzip', None compression_opts: 0-9 for gzip shuffle: True/False

description = 'Keysight X Series Digital Oscilliscope'

allowed_children = [<class 'naqslab_devices.ScopeChannel'>]

generate_code(*hdf5_file*)
Automatically called by compiler to write acquisition instructions to h5 file. Configures counters, analog and digital acquisitions.

acquire(*start_time*)
Call to define time when trigger will happen for scope.

add_device(*device*)

do_checks()

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

minimum_recovery_time = 0

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

trigger(*t*, *duration*)
Request parent trigger device to produce a trigger at time t with given duration.

trigger_edge_type = 'rising'

KeysightXSeries.register_classes

3.7 NovaTechDDS

3.7.1 Overview

This modules covers the 409B-AC, 409B, and 440A DDS frequency synthesizers from Novatech.

Note that the 409B-AC class is largely interchangable with the DDS9m labscript_devices class.

naqslab_devices.NovaTechDDS.blacs_tab
naqslab_devices.NovaTechDDS.blacs_worker
naqslab_devices.NovaTechDDS.labscript_device
naqslab_devices.NovaTechDDS.register_classes
naqslab_devices.NovaTechDDS.runviewer_parser

3.7.2 Detailed Documentation of naqslab_devices.NovaTechDDS

NovaTechDDS.blacs_tab

class naqslab_devices.NovaTechDDS.blacs_tab.**NovaTech409B_ACTab**(**args*, ***kwargs*)

Bases: blacs.device_base_class.DeviceTab

initialise_GUI()

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.NovaTechDDS.blacs_tab.**NovaTech409BTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.NovaTechDDS.blacs_tab.NovaTech409B_ACTab*

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_GUI()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

class naqslab_devices.NovaTechDDS.blacs_tab.**NovaTech440ATab**(**args*, ***kwargs*)

Bases: *naqslab_devices.NovaTechDDS.blacs_tab.NovaTech409B_ACTab*

initialise_GUI()

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

NovaTechDDS.blacs_worker

class naqslab_devices.NovaTechDDS.blacs_worker.**NovaTech409B_ACWorker**(**args*, ***kwargs*)

Bases: blacs.tab_base_classes.Worker

init()
Initialization command run automatically by the BLACS tab on startup. It establishes communication and sends initial default configuration commands

check_connection()
Sends non-command and tests for correct response returns tuple of connection state and reponse string

write_check(*command*)
Sends command and checks and confirms proper execution by reading 'OK' from device.

check_error(*response*)
Parse response for errors and raise appropriate error. If no error, returns response unaltered.

check_remote_values()
Queries device for current output settings. Return results as a dictionary to update the BLACS tab.

program_manual(*front_panel_values*)
Called within the BLACS worker during transitions. This calls program_static for each setting if it isn't already set.

program_static(*channel*, *type*, *value*)
General output parameter programming function. Only sends one command per use.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)

abort_transition_to_buffered()

abort_buffered()

transition_to_manual(*abort=False*)

shutdown()

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

class naqslab_devices.NovaTechDDS.blacs_worker.**NovaTech409BWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.NovaTechDDS.blacs_worker.NovaTech409B_ACWorker*

transition_to_manual(*abort=False*)

abort_buffered()

abort_transition_to_buffered()

check_connection()
Sends non-command and tests for correct response returns tuple of connection state and reponse string

check_error(*response*)
Parse response for errors and raise appropriate error. If no error, returns response unaltered.

check_remote_values()
Queries device for current output settings. Return results as a dictionary to update the BLACS tab.

init()
Initialization command run automatically by the BLACS tab on startup. It establishes communication and sends initial default configuration commands

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Called within the BLACS worker during transitions. This calls program_static for each setting if it isn't already set.

program_static(*channel*, *type*, *value*)
General output parameter programming function. Only sends one command per use.

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)

write_check(*command*)
Sends command and checks and confirms proper execution by reading 'OK' from device.

class naqslab_devices.NovaTechDDS.blacs_worker.**NovaTech440AWorker**(**args*, ***kwargs*)
Bases: *naqslab_devices.NovaTechDDS.blacs_worker.NovaTech409BWorker*

init()
Modified init from 409B-AC. The 440A only supports one baud rate and does not support output mode commands.

program_static(*channel*, *type*, *value*)
General output parameter programming function. Only sends one command per use.

check_remote_values()
The 440A Query command returns values in a different order and does not tell the amplitude.

abort_buffered()

abort_transition_to_buffered()

check_connection()
Sends non-command and tests for correct response returns tuple of connection state and reponse string

check_error(*response*)
Parse response for errors and raise appropriate error. If no error, returns response unaltered.

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Called within the BLACS worker during transitions. This calls program_static for each setting if it isn't already set.

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)

transition_to_manual(*abort=False*)

write_check(*command*)
Sends command and checks and confirms proper execution by reading 'OK' from device.

NovaTechDDS.labscript_device

class naqslab_devices.NovaTechDDS.labscript_device.**NovaTech409B_AC**(*name*, *parent_device*, *com_port=''*, *baud_rate=19200*, *update_mode='synchronous'*, *synchronous_first_line_repeat=Fa*
phase_mode='continuous', *ext_clk=False*, *clk_freq=None*, *clk_mult=None*, *R_option=False*, ***kwargs*)

Bases: labscript.labscript.IntermediateDevice

Labscript device class for NovaTech 409B-AC variant DDS. This device has two dynamic channels (0,1) and two static channels (2,3). If an external clock frequency is enabled, and /R option is not being used, clk_freq (in MHz) and clk_mult (int) must also be defined.

description = 'NT-DDS409B-AC'

allowed_children = [<class 'labscript.labscript.DDS'>, <class 'labscript.labscript.S

clock_limit = 9990

clock_check()
Checks to make sure clk_mult and clk_freq have valid values, as determined by the Novatech documentation. Returns the correct frequency scaling factor to account for different clocking options.

add_device(*device*)

get_default_unit_conversion_classes(*device*)
Child devices call this during their __init__ (with themselves as the argument) to check if there are certain unit calibration classes that they should apply to their outputs, if the user has not otherwise specified a calibration class

quantise_freq(*data*, *device*)
Provides bounds error checking and scales input values to instrument units (0.1 Hz) before ensuring uint32 integer type.

quantise_phase(*data*, *device*)
Ensures phase is wrapped about 360 degrees and scales to instrument units before type casting to uint16.

quantise_amp(*data*, *device*)
Ensures amplitude is within bounds and scales to instrument units (between 0 and 1023) before typecasting to uint16

generate_code(*hdf5_file*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.NovaTechDDS.labscript_device.**NovaTech409B**(*name*, *com_port=''*, *baud_rate=19200*, *phase_mode='default'*, *R_option=False*, *ext_clk=False*, *clk_freq=None*, *clk_mult=None*, ***kwargs*)

Bases: *naqslab_devices.NovaTechDDS.labscript_device.NovaTech409B_AC*

Labscript class for NovaTech 409B DDS. This device has four static DDS output channels.

description = 'NT-DDS409B'

allowed_children = [<class 'labscript.labscript.StaticDDS'>]

clock_limit = 1

generate_code(*hdf5_file*)
Modified version of 409B-AC generate_code that only handles static DDS outputs

add_device(*device*)

clock_check()
Checks to make sure clk_mult and clk_freq have valid values, as determined by the Novatech documentation. Returns the correct frequency scaling factor to account for different clocking options.

get_all_children()

get_all_outputs()

get_default_unit_conversion_classes(*device*)
Child devices call this during their __init__ (with themselves as the argument) to check if there are certain unit calibration classes that they should apply to their outputs, if the user has not otherwise specified a calibration class

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Ensures amplitude is within bounds and scales to instrument units (between 0 and 1023) before typecasting to uint16

quantise_freq(*data*, *device*)
Provides bounds error checking and scales input values to instrument units (0.1 Hz) before ensuring uint32 integer type.

quantise_phase(*data*, *device*)
Ensures phase is wrapped about 360 degrees and scales to instrument units before type casting to uint16.

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

class naqslab_devices.NovaTechDDS.labscript_device.**NovaTech440A**(*name*, *com_port=''*, *baud_rate=19200*, *ext_clk=False*, *clk_freq=None*, ***kwargs*)

Bases: *naqslab_devices.NovaTechDDS.labscript_device.NovaTech409B_AC*

Labscript class for Novatech 440A DDS. This is a high frequency DDS with single channel output that does not support amplitude control

description = 'NT-DDS440A'

allowed_children = [<class 'labscript.labscript.StaticDDS'>]

clock_limit = 1

add_device(*device*)

quantise_freq(*data*, *device*)
Provides bounds error checking and scales input values to instrument units (0.1 Hz) before ensuring uint32 integer type.

generate_code(*hdf5_file*)
Modified generate code from 409B to only accept one channel and ignore amplitude commands which are not supported by the device.

clock_check()
Checks to make sure clk_mult and clk_freq have valid values, as determined by the Novatech documentation. Returns the correct frequency scaling factor to account for different clocking options.

get_all_children()

get_all_outputs()

get_default_unit_conversion_classes(*device*)
Child devices call this during their __init__ (with themselves as the argument) to check if there are certain unit calibration classes that they should apply to their outputs, if the user has not otherwise specified a calibration class

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_amp(*data*, *device*)
Ensures amplitude is within bounds and scales to instrument units (between 0 and 1023) before typecasting to uint16

quantise_phase(*data*, *device*)
Ensures phase is wrapped about 360 degrees and scales to instrument units before type casting to uint16.

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

NovaTechDDS.register_classes

NovaTechDDS.runviewer_parser

class naqslab_devices.NovaTechDDS.runviewer_parser.**NovaTech409B_ACParser**(*path*, *device*)

Bases: object

get_traces(*add_trace*, *clock=None*)

class naqslab_devices.NovaTechDDS.runviewer_parser.**NovaTech409BParser**(*path*, *device*)

Bases: *naqslab_devices.NovaTechDDS.runviewer_parser.NovaTech409B_ACParser*

get_traces(*add_trace*, *clock=None*)

class naqslab_devices.NovaTechDDS.runviewer_parser.**NovaTech440AParser**(*path*, *device*)

Bases: *naqslab_devices.NovaTechDDS.runviewer_parser.NovaTech409B_ACParser*

get_traces(*add_trace*, *clock=None*)

3.8 SR865

3.8.1 Overview

This device class controls the Stanford Research Systems 865 series Lock-in Amplifier. It only implements control functions. Data acquisition is not directly supported at this time. To get buffered data, please use a DAQ connected to the analog outputs of the 865.

naqslab_devices.SR865.blacs_tab
naqslab_devices.SR865.blacs_worker
naqslab_devices.SR865.labscript_device
naqslab_devices.SR865.register_classes

3.8.2 Detailed Documentation of naqslab_devices.SR865

SR865.blacs_tab

class naqslab_devices.SR865.blacs_tab.**SR865Tab**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_tab.VISATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

status_byte_labels = {'bit 0': 'OPC', 'bit 1': 'Input Queue Overflow', 'bit 2':

initialise_GUI()

Loads the standard STBstatus.ui widget and sets the worker defined in __init__

tau_changed(**args*, ***kwargs*)

sens_changed(**args*, ***kwargs*)

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

SR865.blacs_worker

class naqslab_devices.SR865.blacs_worker.**SR865Worker**(**args*, ***kwargs*)
Bases: *naqslab_devices.VISA.blacs_worker.VISAWorker*

program_string = 'OFLT {:d};SCAL {:d};PHAS {:.6f}'

read_string = 'OFLT?;SCAL?;PHAS?'

phase_parser(*phase_string*)
Phase Query string parser

coerce_tau(*tau_constant*)
Returns coerced, valid integer setting. Tau value rounds up. Returns max or min valid setting if out of bound.

coerce_sens(*sensitivity*)
Returns coerced, valid integer setting. Sens value rounds down. Returns max or min valid setting if out of bound.

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

check_remote_values()
Queries the current settings for all three parameters. Parses results to actual numbers and returns.

program_manual(*front_panel_values*)
Performans manual updates from BLACS front panel. Tau and Sensitivity settings are coerced to nearest allowed value

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

check_status()
Queries device state using the ESR register. Bit definitions defined in blacs_tab

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from read_stb

Returns Status byte dictionary as formatted in VISATab

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

SR865.labscript_device

class naqslab_devices.SR865.labscript_device.**SR865**(*name*, *VISA_name*)
Bases: *naqslab_devices.VISA.labscript_device.VISA*

VISA_name can be full VISA connection string or NI-MAX alias

description = 'SR865 Lock-In Amplifier'

allowed_children = None

tau = None

sens = None

phase = None

set_tau(*tau_constant*)
Set the time constant in seconds. Uses numpy digitize to translate to int values. Using digitize corrects for round-off errors and coerces input to nearest allowed setting.

set_sens(*sensitivity*)
Set the sensitivity in Volts Uses numpy digitize to translate to int values. Using digitize corrects for round-off errors and coerces input to nearest allowed setting.

set_phase(*phase*)
Set the phase reference in degrees Device auto-converts to -180,180 range

generate_code(*hdf5_file*)
Generates the transition to buffered code in the h5 file. If parameter is not specified in shot, NaN and -1 values are set to tell worker not to change the value when programming.

add_device(*device*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

SR865.register_classes

3.9 TektronixTDS

3.9.1 Overview

This covers the TDS series of Tektronix oscilloscopes.

naqslab_devices.TektronixTDS.blacs_tab
naqslab_devices.TektronixTDS.blacs_worker
naqslab_devices.TektronixTDS.labscript_device
naqslab_devices.TektronixTDS.register_classes

3.9.2 Detailed Documentation of naqslab_devices.TektronixTDS

TektronixTDS.blacs_tab

class naqslab_devices.TektronixTDS.blacs_tab.**TDS_ScopeTab**(**args*, ***kwargs*)
Bases: *naqslab_devices.VISA.blacs_tab.VISATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

status_byte_labels = {'bit 0': 'Unused', 'bit 1': 'Unused', 'bit 2': 'Query Erro

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

TektronixTDS.blacs_worker

class naqslab_devices.TektronixTDS.blacs_worker.**TDS_ScopeWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_worker.VISAWorker*

setup_string = ':HEADER OFF;*ESE 60;*SRE 32;*CLS;:DAT:ENC RIB;WID 2;'

read_y_parameters_string = ':DAT:SOU CH%d;:WFMPRE:YZE?;YMU?;YOFF?'

read_x_parameters_string = ':WFMPRE:XZE?;XIN?'

read_waveform_string = 'CURV?'

waveform_parser(*raw_waveform_array*, *y0*, *dy*, *yoffset*)
Parses the numpy array from the CURV? query.

init()
Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

check_status()
Uses the more informative ESR register.

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

check_remote_values()

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

program_manual(*front_panel_values*)
Over-ride this method if remote programming is supported.

Returns `VISAWorker.check_remote_values()`

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)
Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file
- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

TektronixTDS.labscript_device

class naqslab_devices.TektronixTDS.labscript_device.**TDS_Scope**(*name*, *VISA_name*, *trigger_device*, *trigger_connection*, ***kwargs*)

Bases: labscript.labscript.TriggerableDevice

VISA_name can be full VISA connection string or NI-MAX alias. Trigger Device should be fast clocked device.

description = 'Tektronics TDS Series Digital Oscilliscope'

allowed_children = [<class 'naqslab_devices.ScopeChannel'>]

trigger_duration = 0.001

generate_code(*hdf5_file*)

acquire(*start_time*)
Call to define time when trigger will happen for scope.

add_device(*device*)

do_checks()

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

minimum_recovery_time = 0

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, ...} where each val is a list [var1, var2, ...] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

trigger(*t*, *duration*)
Request parent trigger device to produce a trigger at time t with given duration.

trigger_edge_type = 'rising'

TektronixTDS.register_classes

3.10 KeysightDCSupply

3.10.1 Overview

This device class controls Keysight/Agilent DC Power Supplies.

The underlying implementation is to use the outputs as StaticAnalogOut. The type of analog quantity (voltage or current source) is configured at the device level based on whether the outputs are voltage limited or current limited.

At present, this driver is fairly limited in its control. Most importantly, the driver does not set both voltage and current limits (i.e. setting a max current limit when in constant voltage mode). A modified StaticAnalogOut will be necessary. It is also only tested for the E364x series single output supplies. It is written such that multiple outputs could be supported easily, but that functionality is untested.

naqslab_devices.KeysightDCSupply.blacs_tab
naqslab_devices.KeysightDCSupply.blacs_worker
naqslab_devices.KeysightDCSupply.labscript_device
naqslab_devices.KeysightDCSupply.register_classes

3.10.2 Detailed Documentation of naqslab_devices.KeysightDCSupply

KeysightDCSupply.blacs_tab

class naqslab_devices.KeysightDCSupply.blacs_tab.**KeysightDCSupplyTab**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_tab.VISATab*

You MUST override this method in order to define the device worker. You then call this parent method to finish initialization.

status_byte_labels = {'bit 0': 'Constant Current Mode', 'bit 1': 'Constant Voltage

initialise_GUI()
Loads the standard STBstatus.ui widget and sets the worker defined in __init__

ICON_BUSY = ':/qtutils/fugue/hourglass'

ICON_ERROR = ':/qtutils/fugue/exclamation'

ICON_FATAL_ERROR = ':/qtutils/fugue/exclamation-red'

ICON_OK = ':/qtutils/fugue/tick'

STBui_path = 'C:\\labscript_suite\\naqslab_devices\\VISA\\STBstatus.ui'

abort_buffered(**args*, ***kwargs*)

abort_transition_to_buffered(**args*, ***kwargs*)

add_secondary_worker(*worker*)

auto_create_widgets()

auto_place_widgets(**args*)

check_remote_values(**args*, ***kwargs*)

check_time()

clean_ui_on_restart()

close_tab(*finalise=True*)
Close the tab, terminate subprocesses and join the mainloop thread. If finalise=False, then do not terminate subprocesses or join the mainloop. In this case, callers must manually call finalise_close_tab() to perform these potentially blocking operations

connect_restart_receiver(*function*)

continue_restart(*currentpage*)
Called in a thread for the stages of restarting that may be blocking, so as to not block the main thread. Calls subsequent GUI operations in the main thread once finished blocking.

create_analog_outputs(*analog_properties*)

create_analog_widgets(*channel_properties*)

create_dds_outputs(*dds_properties*)

create_dds_widgets(*channel_properties*)

create_device_properties(*device_properties*)

create_digital_outputs(*digital_properties*)

create_digital_widgets(*channel_properties*)

create_image_outputs(*image_properties*)

create_image_widgets(*channel_properties*)

create_property_widgets(*device_properties*)

create_worker(*name*, *WorkerClass*, *workerargs=None*)
Set up a worker process. WorkerClass can either be a subclass of Worker, or a string containing a fully qualified import path to a worker. The latter is useful if the worker class is in a separate file with global imports or other import-time behaviour that is undesirable to have run in the main process, for example if the imports may not be available to the main process (as may be the case once remote worker processes are implemented and the worker may be on a separate computer). The worker process will not be started immediately, it will be started once the state machine mainloop begins running. This way errors in startup will be handled using the normal state machine machinery.

property device_name

disconnect_restart_receiver(*function*)

property error_message

finalise_close_tab(*currentpage*)

finalise_restart(*currentpage*)

property force_full_buffered_reprogram

get_all_save_data()

get_builtin_save_data()
Get builtin settings to be restored like whether the terminal is visible. Not to be overridden.

get_channel(*channel*)

get_child_from_connection_table(*parent_device_name*, *port*)

get_front_panel_properties()

get_front_panel_values()

get_property(*property*)

get_save_data()

get_tab_layout()

hide_error()

initialise_workers()

mainloop()

property mode

on_force_full_buffered_reprogram()

on_resolve_value_inconsistency()

property primary_worker

program_device(**args*, ***kwargs*)

program_device_properties(**args*, ***kwargs*)

queue_work(*worker_process*, *worker_function*, **args*, ***kwargs*)

restart(**args*)

restore_builtin_save_data(*data*)
Restore builtin settings to be restored like whether the terminal is visible. Not to be overridden.

restore_save_data(*data*)

send_clear(**args*, ***kwargs*)

set_tab_icon_and_colour()
Set the tab icon and the colour of its text to the values of self._tab_icon and self._tab_text_colour respectively

set_terminal_visible(*visible*)

shutdown_workers(**args*, ***kwargs*)

start_run(**args*, ***kwargs*)

property state

statemachine_timeout_add(*delay*, *statefunction*, **args*, ***kwargs*)

statemachine_timeout_remove(*statefunction*)

statemachine_timeout_remove_all()

status_monitor(**args*, ***kwargs*)

status_widget = 'STBstatus.ui'

supports_remote_value_check(*support*)

supports_smart_programming(*support*)

transition_to_buffered(**args*, ***kwargs*)

transition_to_manual(**args*, ***kwargs*)

update_from_settings(*settings*)

KeysightDCSupply.blacs_worker

class naqslab_devices.KeysightDCSupply.blacs_worker.**KeysightDCSupplyWorker**(**args*, ***kwargs*)

Bases: *naqslab_devices.VISA.blacs_worker.VISAWorker*

esr_mask = 60

qsr_mask = 1539

init_string = '*ESE 60;STAT:QUES:ENAB 1539;*CLS'

ident_string = 'E364'

write_both_string = 'APPL %.5f, %.5f'

write_volt_string = 'VOLT %.5f'

write_current_string = 'CURR %.5f'

read_string = 'APPL?'

read_parser(*response*)

Parses the Voltage & Amplitude response string

Parameters **response** (*str*) – Instrument response to current voltage/current query. Has format of "d.ddddd, d.ddddd"

Returns

containing

V (float): Current Voltage Setting A (float): Current Current Setting

Return type (tuple)

init()

Initializes basic worker and opens VISA connection to device.

Default connection timeout is 2 seconds

check_remote_values()

program_manual(*front_panel_values*)

Over-ride this method if remote programming is supported.

Returns VISAWorker.check_remote_values()

transition_to_buffered(*device_name*, *h5file*, *initial_values*, *fresh*)

Stores various device handles for use in transition_to_manual method.

Automatically called by BLACS. Should be over-ridden by inheritors.

Parameters

- **device_name** (*str*) – Name of device from connectiontable
- **h5file** (*str*) – path to shot h5_file

- **initial_values** (*dict*) – Contains the start of shot values
- **fresh** (*bool*) – Indicates if smart_programming should be refreshed this shot

check_status()
Customised check status for Keysight DC supplies.

This method combines flags from the event status register and the questionable status register.

merge_registers(*esr*, *qsr*, *cond*)
Merges the ESR & QSR registers with the condition into a hybrid register and converts to a dictionary for display on the BLACStab.

Parameters

- **esr** (*int*) – Value of the Event Status Register
- **qsr** (*int*) – Value of the Questionable Status Register
- **cond** (*int*) – Response of STATus:QUEStionable:CONDition?

Returns Dictionary of values from the esr and qsr registers.

Return type return_vals (dict)

abort_buffered()
Special abort shot code belongs here.

abort_transition_to_buffered()
Special abort shot configuration code belongs here.

clear(*value*)
Sends standard *CLR to clear registers of device.

Parameters **value** (*bool*) – value of Clear button in STBstatus.ui widget

convert_register(*register*)
Converts returned register value to dict of bools

Parameters **register** (*int*) – Status register value returned from `read_stb`

Returns Status byte dictionary as formatted in `VISATab`

Return type dict

interrupt_startup(*reason='Process.interrupt_startup() called'*)
Called from the parent process. Interrupt all blocking operations on starting the child process, causing Process.start() to raise Interrupted(reason). After interruption, self.child may be None if startup was interrupted before the child was started, otherwise self.child will be the child Popen object, which could be at any stage of setting up its connection with the parent. This method may be called multiple times without raising an exception, it will simply do nothing if startup has previously been interrupted

mainloop()

run(*worker_name*, *device_name*, *extraargs*)
The method that gets called in the subprocess. To be overridden by subclasses

shutdown()
Closes VISA connection to device.

start(**args*, ***kwargs*)
Call in the parent process to start a subprocess. Passes args and kwargs to the run() method

terminate(*wait_timeout=None*, ***kwargs*)
Interrupt process startup if not already done, ensuring self.child exists or is None if startup was interrupted before the process was created. Then if the child is not None, call Popen.terminate() and Popen.wait() on it.

transition_to_manual(*abort=False*)
Simple transition_to_manual method where no data is saved.

KeysightDCSupply.labscript_device

class naqslab_devices.KeysightDCSupply.labscript_device.**KeysightDCSupply**(*name*, *VISA_name*, *range='LOW'*, *volt_limits=(0, 1)*, *current_limits=(0, 1)*, *limited='volt'*)

Bases: *naqslab_devices.VISA.labscript_device.VISA*

Keysight DC Power Supply

The labscript_device for Keysight DC Power supplies. Currently only tested for E364xA single output series devices.

Parameters

- **name** (*str*) – labscript name to assign to device. Must be an allowed python variable name.
- **VISA_name** (*str*) – VISA connection string to device. Can be alias configured in NI-MAX.
- **range** (*str*) – configures which voltage range to use. Default is 'LOW'.
- **volt_limits** (*iterable*) – voltage limits, in volts
- **current_limits** (*iterable*) – current limits, in amps
- **limited** (*str*) – Sets whether output is configured to be voltage or current limited. Default is 'volt'

description = 'DC Power Supply'

allowed_children = [<class 'labscript.labscript.StaticAnalogOut'>]

allowed_outputs = [0]

quantise_volt(*data*, *output*)
Quantize the currents in units of V and check it's within bounds

quantise_current(*data*, *output*)
Quantize the currents in units of A and check it's within bounds

generate_code(*hdf5_file*)
Method to generate instructions for blacs_worker to program device.

Must be over-ridden.

add_device(*device*)

get_all_children()

get_all_outputs()

get_properties(*location=None*)
Get all properties in location

If location is None we return all keys

get_property(*name*, *location=None*, **args*, ***kwargs*)

init_device_group(*hdf5_file*)

property parent_clock_line

property pseudoclock_device

quantise_to_pseudoclock(*times*)

set_properties(*properties_dict*, *property_names*, *overwrite=False*)
Add one or a bunch of properties packed into properties_dict

property_names is a dictionary {key:val, . . . } where each val is a list [var1, var2, . . .] of variables to be pulled from properties_dict and added to the property with name key (it's location)

set_property(*name*, *value*, *location=None*, *overwrite=False*)

property t0
The earliest time output can be commanded from this device at the start of the experiment. This is nonzeo on secondary pseudoclock devices due to triggering delays.

KeysightDCSupply.register_classes

CHAPTER

FOUR

BUILDING DOCUMENTATION

The API documentation for this library leverages the Sphinx automatic python documenation generator. The general structure is to use the Sphinx-apidoc infrastucture to read source code docstrings to automatically build the function/class reference documentation for each device. Hand-written ReStructedText files (quick syntax guide) are then used to provide high-level documentation that imports the auto-generated documentation. A makefile is provided to run the correct `apidoc` and `sphinx-build` commands.

4.1 Sphinx Environment

In order to build the documentation from scratch, a functioning `labscript` environment is needed with sphinx also installed. Because sphinx require a large amount of dependencies, it is recommended to copy your local `labscript` environment then install and use sphinx from there.

Sphinx can be installed into the `labscript` environment by installing the packages: `sphinx>=2.2` and `sphinx_rtd_theme`. This will allow building of the html documentation.

If you also wish to build the pdf documentation, you must also install `perl` and a latex environment (such as MikTeX for windows). The latex build is controlled using the `latexmk` latex package and requires a great many other latex packages. A partial list of required latex packages is: cmap, fncychap, tabulary, parskip, capt-of.

4.2 Sphinx Build

The documentation build is automated through makefiles. All commands are run from the `doc` subfolder.

The automated documentation build is perfomed using

```
make apidoc
```

The auto-generated files are placed in the `_apidoc` subfolder of `doc` and are given the import name of the module with the file suffix `.inc`.

The complete documentation is built using `sphinx-build` and currently supports two targets: html and latex-pdf. They are run using

```
make html
#or
make latexpdf
```

4.3 Adding Documentation

The main documentation tree is found in `index.rst`. New devices should be added using `devices.rst`. Each device should have it's own rst file that provides some high-level documentation and includes the auto-generated apidoc file. All modules and submodules in the top-level directory of the library will have documenta-

tion auto-generated using apidoc. It is up to the user to include those files where appropriate in the documenation. The include statement is of the form:

```
.. include:: _apidoc\naqslab_devices.NovaTechDDS.inc
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

SIX

CREDITS

Authors David Meyer Zac Castillo

Licence Simplified BSD License

Version 0.2.7

PYTHON MODULE INDEX

n

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W