gnuradio.qtgui

Provides a GUI interface using the QT backend.

`gnuradio.qtgui.`**`ber_sink_b`**(*pmt_vector_float esnos, int curves=1, int berminerrors=100, float berLimit=-7.0, std::vector< std::string, std::allocator< std::string > > curvenames, QWidget * parent=None*) → ber_sink_b_sptr

Constructor Specific Documentation:

Parameters:
- **esnos** –
- **curves** –
- **berminerrors** –
- **berLimit** –
- **curvenames** –
- **parent** –

`ber_sink_b_sptr.`**`active_thread_priority`**(*ber_sink_b_sptr self*) → int

`ber_sink_b_sptr.`**`d_qApplication`**(*ber_sink_b_sptr self*) → QApplication *

`ber_sink_b_sptr.`**`declare_sample_delay`**(*ber_sink_b_sptr self, int which, int delay*)

declare_sample_delay(ber_sink_b_sptr self, unsigned int delay)

`ber_sink_b_sptr.`**`enable_autoscale`**(*ber_sink_b_sptr self, bool en*)

`ber_sink_b_sptr.`**`enable_menu`**(*ber_sink_b_sptr self, bool en=True*)

`ber_sink_b_sptr.`**`exec_`**(*ber_sink_b_sptr self*)

`ber_sink_b_sptr.`**`line_alpha`**(*ber_sink_b_sptr self, int which*) → double

`ber_sink_b_sptr.`**`line_color`**(*ber_sink_b_sptr self, int which*) → std::string

`ber_sink_b_sptr.`**`line_label`**(*ber_sink_b_sptr self, int which*) → std::string

`ber_sink_b_sptr.`**`line_marker`**(*ber_sink_b_sptr self, int which*) → int

`ber_sink_b_sptr.`**`line_style`**(*ber_sink_b_sptr self, int which*) → int

`ber_sink_b_sptr.`**`line_width`**(*ber_sink_b_sptr self, int which*) → int

`ber_sink_b_sptr.`**`message_subscribers`**(*ber_sink_b_sptr self, swig_int_ptr which_port*) → swig_int_ptr

`ber_sink_b_sptr.`**`min_noutput_items`**(*ber_sink_b_sptr self*) → int

`ber_sink_b_sptr.`**`nsamps`**(*ber_sink_b_sptr self*) → int

`ber_sink_b_sptr.`**`pc_input_buffers_full_avg`**(*ber_sink_b_sptr self, int which*) → float

pc_input_buffers_full_avg(ber_sink_b_sptr self) -> pmt_vector_float

`ber_sink_b_sptr.`**`pc_noutput_items_avg`**(*ber_sink_b_sptr self*) → float

`ber_sink_b_sptr.`**`pc_nproduced_avg`**(*ber_sink_b_sptr self*) → float

`ber_sink_b_sptr.`**`pc_output_buffers_full_avg`**(*ber_sink_b_sptr self, int which*) → float

pc_output_buffers_full_avg(ber_sink_b_sptr self) -> pmt_vector_float

`ber_sink_b_sptr.`**`pc_throughput_avg`**(*ber_sink_b_sptr self*) → float ber_sink_b_sptr.**pc_work_time_avg**(*ber_sink_b_sptr self*) → float

ber_sink_b_sptr.**pc_work_time_total**(*ber_sink_b_sptr self*) → float

ber_sink_b_sptr.**pyqwidget**(*ber_sink_b_sptr self*) → PyObject *

ber_sink_b_sptr.**sample_delay**(*ber_sink_b_sptr self*, *int which*) → unsigned int

ber_sink_b_sptr.**set_line_alpha**(*ber_sink_b_sptr self*, *int which*, *double alpha*)

ber_sink_b_sptr.**set_line_color**(*ber_sink_b_sptr self*, *int which*, *std::string const & color*)

ber_sink_b_sptr.**set_line_label**(*ber_sink_b_sptr self*, *int which*, *std::string const & label*)

ber_sink_b_sptr.**set_line_marker**(*ber_sink_b_sptr self*, *int which*, *int marker*)

ber_sink_b_sptr.**set_line_style**(*ber_sink_b_sptr self*, *int which*, *int style*)

ber_sink_b_sptr.**set_line_width**(*ber_sink_b_sptr self*, *int which*, *int width*)

ber_sink_b_sptr.**set_min_noutput_items**(*ber_sink_b_sptr self*, *int m*)

ber_sink_b_sptr.**set_size**(*ber_sink_b_sptr self*, *int width*, *int height*)

ber_sink_b_sptr.**set_thread_priority**(*ber_sink_b_sptr self*, *int priority*) → int

ber_sink_b_sptr.**set_title**(*ber_sink_b_sptr self*, *std::string const & title*)

ber_sink_b_sptr.**set_update_time**(*ber_sink_b_sptr self*, *double t*)

ber_sink_b_sptr.**set_x_axis**(*ber_sink_b_sptr self*, *double min*, *double max*)

ber_sink_b_sptr.**set_y_axis**(*ber_sink_b_sptr self*, *double min*, *double max*)

ber_sink_b_sptr.**thread_priority**(*ber_sink_b_sptr self*) → int

ber_sink_b_sptr.**title**(*ber_sink_b_sptr self*) → std::string

gnuradio.qtgui.**const_sink_c**(*int size*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → const_sink_c_sptr

A graphical sink to display the IQ constellation of multiple signals.

This is a QT-based graphical sink the takes set of a complex streams and plots them on an IQ constellation plot.

The sink supports plotting streaming complex data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Complex Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of complex 32 values (pmt::is_c32vector) or PDUs where the data is a uniform vector of complex 32 values.

Constructor Specific Documentation:

Build a constellation plot sink.

Parameters:
- **size** – number of points to plot at once
- **name** – title for the plot
- **nconnections** – number of signals connected to sink
- **parent** – a QWidget parent object, if any

const_sink_c_sptr.**active_thread_priority**(*const_sink_c_sptr self*) → int const_sink_c_sptr.**d_qApplication**(*const_sink_c_sptr self*) → QApplication *

const_sink_c_sptr.**declare_sample_delay**(*const_sink_c_sptr self*, *int which*, *int delay*)
declare_sample_delay(const_sink_c_sptr self, unsigned int delay)

const_sink_c_sptr.**disable_legend**(*const_sink_c_sptr self*)

const_sink_c_sptr.**enable_autoscale**(*const_sink_c_sptr self*, *bool en*)

const_sink_c_sptr.**enable_axis_labels**(*const_sink_c_sptr self*, *bool en=True*)

const_sink_c_sptr.**enable_grid**(*const_sink_c_sptr self*, *bool en*)

const_sink_c_sptr.**enable_menu**(*const_sink_c_sptr self*, *bool en=True*)

const_sink_c_sptr.**exec_**(*const_sink_c_sptr self*)

const_sink_c_sptr.**line_alpha**(*const_sink_c_sptr self*, *int which*) → double

const_sink_c_sptr.**line_color**(*const_sink_c_sptr self*, *int which*) → std::string

const_sink_c_sptr.**line_label**(*const_sink_c_sptr self*, *int which*) → std::string

const_sink_c_sptr.**line_marker**(*const_sink_c_sptr self*, *int which*) → int

const_sink_c_sptr.**line_style**(*const_sink_c_sptr self*, *int which*) → int

const_sink_c_sptr.**line_width**(*const_sink_c_sptr self*, *int which*) → int

const_sink_c_sptr.**message_subscribers**(*const_sink_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

const_sink_c_sptr.**min_noutput_items**(*const_sink_c_sptr self*) → int

const_sink_c_sptr.**nsamps**(*const_sink_c_sptr self*) → int

const_sink_c_sptr.**pc_input_buffers_full_avg**(*const_sink_c_sptr self*, *int which*) → float
pc_input_buffers_full_avg(const_sink_c_sptr self) -> pmt_vector_float

const_sink_c_sptr.**pc_noutput_items_avg**(*const_sink_c_sptr self*) → float

const_sink_c_sptr.**pc_nproduced_avg**(*const_sink_c_sptr self*) → float

const_sink_c_sptr.**pc_output_buffers_full_avg**(*const_sink_c_sptr self*, *int which*) → float
pc_output_buffers_full_avg(const_sink_c_sptr self) -> pmt_vector_float

const_sink_c_sptr.**pc_throughput_avg**(*const_sink_c_sptr self*) → float

const_sink_c_sptr.**pc_work_time_avg**(*const_sink_c_sptr self*) → float

const_sink_c_sptr.**pc_work_time_total**(*const_sink_c_sptr self*) → float

const_sink_c_sptr.**pyqwidget**(*const_sink_c_sptr self*) → PyObject *

const_sink_c_sptr.**qwidget**(*const_sink_c_sptr self*) → QWidget *

const_sink_c_sptr.**reset**(*const_sink_c_sptr self*)

const_sink_c_sptr.**sample_delay**(*const_sink_c_sptr self*, *int which*) → unsigned int const_sink_c_sptr.**set_line_alpha**(*const_sink_c_sptr self*, *int which*, *double alpha*)

const_sink_c_sptr.**set_line_color**(*const_sink_c_sptr self*, *int which*, *std::string const & color*)

const_sink_c_sptr.**set_line_label**(*const_sink_c_sptr self*, *int which*, *std::string const & label*)

const_sink_c_sptr.**set_line_marker**(*const_sink_c_sptr self*, *int which*, *int marker*)

const_sink_c_sptr.**set_line_style**(*const_sink_c_sptr self*, *int which*, *int style*)

const_sink_c_sptr.**set_line_width**(*const_sink_c_sptr self*, *int which*, *int width*)

const_sink_c_sptr.**set_min_noutput_items**(*const_sink_c_sptr self*, *int m*)

const_sink_c_sptr.**set_nsamps**(*const_sink_c_sptr self*, *int const newsize*)

const_sink_c_sptr.**set_size**(*const_sink_c_sptr self*, *int width*, *int height*)

const_sink_c_sptr.**set_thread_priority**(*const_sink_c_sptr self*, *int priority*) → int

const_sink_c_sptr.**set_title**(*const_sink_c_sptr self*, *std::string const & title*)

const_sink_c_sptr.**set_trigger_mode**(*const_sink_c_sptr self*, *gr::qtgui::trigger_mode mode*, *gr::qtgui::trigger_slope slope*, *float level*, *int channel*, *std::string const & tag_key*)

Set up a trigger for the sink to know when to start plotting. Useful to isolate events and avoid noise.

The trigger modes are Free, Auto, Normal, and Tag (see gr::qtgui::trigger_mode). The first three are like a normal oscope trigger function. Free means free running with no trigger, auto will trigger if the trigger event is seen, but will still plot otherwise, and normal will hold until the trigger event is observed. The Tag trigger mode allows us to trigger off a specific stream tag. The tag trigger is based only on the name of the tag, so when a tag of the given name is seen, the trigger is activated.

In auto and normal mode, we look for the slope of the magnitude of the signal. As a constellation sink, this only takes in complex numbers to plot. Given a gr::qtgui::trigger_slope as either Positive or Negative, if the magnitude between two samples moves in the given direction (x[1] > x[0] for Positive or x[1] < x[0] for Negative), then the trigger is activated.

const_sink_c_sptr.**set_update_time**(*const_sink_c_sptr self*, *double t*)

const_sink_c_sptr.**set_x_axis**(*const_sink_c_sptr self*, *double min*, *double max*)

const_sink_c_sptr.**set_y_axis**(*const_sink_c_sptr self*, *double min*, *double max*)

const_sink_c_sptr.**thread_priority**(*const_sink_c_sptr self*) → int

const_sink_c_sptr.**title**(*const_sink_c_sptr self*) → std::string

gnuradio.qtgui.**freq_sink_c**(*int fftsize*, *int wintype*, *double fc*, *double bw*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → freq_sink_c_sptr

A graphical sink to display multiple signals in frequency.

This is a QT-based graphical sink the takes set of a complex streams and plots the PSD. Each signal is plotted with a different color, and theand functions can be used to change the lable and color for a given input number.

The sink supports plotting streaming complex data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Complex Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of complex 32 values (pmt::is_c32vector) or PDUs where the data is a uniform vector of complex 32 values.

Message Ports:

Constructor Specific Documentation:

Build a complex PSD sink.

Parameters:

- **fftsize** – size of the FFT to compute and display. If using the PDU message port to plot samples, the length of each PDU must be a multiple of the FFT size.
- **wintype** – type of window to apply (see gr::fft::window::win_type)
- **fc** – center frequency of signal (use for x-axis labels)
- **bw** – bandwidth of signal (used to set x-axis labels)
- **name** – title for the plot
- **nconnections** – number of signals to be connected to the sink. The PDU message port is always available for a connection, and this value must be set to 0 if only the PDU message port is being used.
- **parent** – a QWidget parent object, if any

`freq_sink_c_sptr.`**`active_thread_priority`**(*freq_sink_c_sptr self*) → int

`freq_sink_c_sptr.`**`clear_max_hold`**(*freq_sink_c_sptr self*)

`freq_sink_c_sptr.`**`clear_min_hold`**(*freq_sink_c_sptr self*)

`freq_sink_c_sptr.`**`d_qApplication`**(*freq_sink_c_sptr self*) → QApplication *

`freq_sink_c_sptr.`**`declare_sample_delay`**(*freq_sink_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(freq_sink_c_sptr self, unsigned int delay)

`freq_sink_c_sptr.`**`disable_legend`**(*freq_sink_c_sptr self*)

`freq_sink_c_sptr.`**`enable_autoscale`**(*freq_sink_c_sptr self*, *bool en=True*)

`freq_sink_c_sptr.`**`enable_axis_labels`**(*freq_sink_c_sptr self*, *bool en=True*)

`freq_sink_c_sptr.`**`enable_control_panel`**(*freq_sink_c_sptr self*, *bool en=True*)

`freq_sink_c_sptr.`**`enable_grid`**(*freq_sink_c_sptr self*, *bool en=True*)

`freq_sink_c_sptr.`**`enable_max_hold`**(*freq_sink_c_sptr self*, *bool en*)

`freq_sink_c_sptr.`**`enable_menu`**(*freq_sink_c_sptr self*, *bool en=True*)

`freq_sink_c_sptr.`**`enable_min_hold`**(*freq_sink_c_sptr self*, *bool en*)

`freq_sink_c_sptr.`**`exec_`**(*freq_sink_c_sptr self*)

`freq_sink_c_sptr.`**`fft_average`**(*freq_sink_c_sptr self*) → float

`freq_sink_c_sptr.`**`fft_size`**(*freq_sink_c_sptr self*) → int

`freq_sink_c_sptr.`**`fft_window`**(*freq_sink_c_sptr self*) → gr::filter::firdes::win_type

`freq_sink_c_sptr.`**`line_alpha`**(*freq_sink_c_sptr self*, *int which*) → double freq_sink_c_sptr.**line_color**(*freq_sink_c_sptr self*, *int which*) → std::string

freq_sink_c_sptr.**line_label**(*freq_sink_c_sptr self*, *int which*) → std::string

freq_sink_c_sptr.**line_marker**(*freq_sink_c_sptr self*, *int which*) → int

freq_sink_c_sptr.**line_style**(*freq_sink_c_sptr self*, *int which*) → int

freq_sink_c_sptr.**line_width**(*freq_sink_c_sptr self*, *int which*) → int

freq_sink_c_sptr.**message_subscribers**(*freq_sink_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

freq_sink_c_sptr.**min_noutput_items**(*freq_sink_c_sptr self*) → int

freq_sink_c_sptr.**pc_input_buffers_full_avg**(*freq_sink_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(freq_sink_c_sptr self) -> pmt_vector_float

freq_sink_c_sptr.**pc_noutput_items_avg**(*freq_sink_c_sptr self*) → float

freq_sink_c_sptr.**pc_nproduced_avg**(*freq_sink_c_sptr self*) → float

freq_sink_c_sptr.**pc_output_buffers_full_avg**(*freq_sink_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(freq_sink_c_sptr self) -> pmt_vector_float

freq_sink_c_sptr.**pc_throughput_avg**(*freq_sink_c_sptr self*) → float

freq_sink_c_sptr.**pc_work_time_avg**(*freq_sink_c_sptr self*) → float

freq_sink_c_sptr.**pc_work_time_total**(*freq_sink_c_sptr self*) → float

freq_sink_c_sptr.**pyqwidget**(*freq_sink_c_sptr self*) → PyObject *

freq_sink_c_sptr.**qwidget**(*freq_sink_c_sptr self*) → QWidget *

freq_sink_c_sptr.**reset**(*freq_sink_c_sptr self*)

freq_sink_c_sptr.**sample_delay**(*freq_sink_c_sptr self*, *int which*) → unsigned int

freq_sink_c_sptr.**set_fft_average**(*freq_sink_c_sptr self*, *float const fftavg*)

freq_sink_c_sptr.**set_fft_size**(*freq_sink_c_sptr self*, *int const fftsize*)

freq_sink_c_sptr.**set_fft_window**(*freq_sink_c_sptr self*, *gr::filter::firdes::win_type const win*)

freq_sink_c_sptr.**set_frequency_range**(*freq_sink_c_sptr self*, *double const centerfreq*, *double const bandwidth*)

freq_sink_c_sptr.**set_line_alpha**(*freq_sink_c_sptr self*, *int which*, *double alpha*)

freq_sink_c_sptr.**set_line_color**(*freq_sink_c_sptr self*, *int which*, *std::string const & color*)

freq_sink_c_sptr.**set_line_label**(*freq_sink_c_sptr self*, *int which*, *std::string const & label*)

freq_sink_c_sptr.**set_line_marker**(*freq_sink_c_sptr self*, *int which*, *int marker*)

freq_sink_c_sptr.**set_line_style**(*freq_sink_c_sptr self*, *int which*, *int style*)

freq_sink_c_sptr.**set_line_width**(*freq_sink_c_sptr self*, *int which*, *int width*) freq_sink_c_sptr.**set_min_noutput_items**(*freq_sink_c_sptr self*, *int m*)

freq_sink_c_sptr.**set_size**(*freq_sink_c_sptr self*, *int width*, *int height*)

freq_sink_c_sptr.**set_thread_priority**(*freq_sink_c_sptr self*, *int priority*) → int

freq_sink_c_sptr.**set_title**(*freq_sink_c_sptr self*, *std::string const & title*)

freq_sink_c_sptr.**set_trigger_mode**(*freq_sink_c_sptr self*, *gr::qtgui::trigger_mode mode*, *float level*, *int channel*, *std::string const & tag_key*)

Set up a trigger for the sink to know when to start plotting. Useful to isolate events and avoid noise.

The trigger modes are Free, Auto, Normal, and Tag (see gr::qtgui::trigger_mode). The first three are like a normal trigger function. Free means free running with no trigger, auto will trigger if the trigger event is seen, but will still plot otherwise, and normal will hold until the trigger event is observed. The Tag trigger mode allows us to trigger off a specific stream tag. The tag trigger is based only on the name of the tag, so when a tag of the given name is seen, the trigger is activated.

In auto and normal mode, we look to see if the magnitude of the any FFT point is over the set level.

freq_sink_c_sptr.**set_update_time**(*freq_sink_c_sptr self*, *double t*)

freq_sink_c_sptr.**set_y_axis**(*freq_sink_c_sptr self*, *double min*, *double max*)

freq_sink_c_sptr.**set_y_label**(*freq_sink_c_sptr self*, *std::string const & label*, *std::string const & unit*)

freq_sink_c_sptr.**thread_priority**(*freq_sink_c_sptr self*) → int

freq_sink_c_sptr.**title**(*freq_sink_c_sptr self*) → std::string

gnuradio.qtgui.**freq_sink_f**(*int fftsize*, *int wintype*, *double fc*, *double bw*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → freq_sink_f_sptr

A graphical sink to display multiple signals in frequency.

This is a QT-based graphical sink the takes set of a floating point streams and plots the PSD. Each signal is plotted with a different color, and theand functions can be used to change the lable and color for a given input number.

The sink supports plotting streaming float data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Float Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of float 32 values (pmt::is_f32vector) or PDUs where the data is a uniform vector of float 32 values.

Message Ports:

Constructor Specific Documentation:

Build a floating point PSD sink.

Parameters:

- **fftsize** – size of the FFT to compute and display. If using the PDU message port to plot samples, the length of each PDU must be a multiple of the FFT size.
- **wintype** – type of window to apply (see gr::fft::window::win_type)
- **fc** – center frequency of signal (use for x-axis labels)
- **bw** – bandwidth of signal (used to set x-axis labels)
- **name** – title for the plot
- **nconnections** – number of signals to be connected to the sink. The PDU message port is always available for a connection, and this value must be set to 0 if only the PDU message port is being used.
- **parent** – a QWidget parent object, if any

`freq_sink_f_sptr.`**`active_thread_priority`**(*freq_sink_f_sptr self*) → int

`freq_sink_f_sptr.`**`clear_max_hold`**(*freq_sink_f_sptr self*)

`freq_sink_f_sptr.`**`clear_min_hold`**(*freq_sink_f_sptr self*)

`freq_sink_f_sptr.`**`d_qApplication`**(*freq_sink_f_sptr self*) → QApplication *

`freq_sink_f_sptr.`**`declare_sample_delay`**(*freq_sink_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(freq_sink_f_sptr self, unsigned int delay)

`freq_sink_f_sptr.`**`disable_legend`**(*freq_sink_f_sptr self*)

`freq_sink_f_sptr.`**`enable_autoscale`**(*freq_sink_f_sptr self*, *bool en=True*)

`freq_sink_f_sptr.`**`enable_axis_labels`**(*freq_sink_f_sptr self*, *bool en=True*)

`freq_sink_f_sptr.`**`enable_control_panel`**(*freq_sink_f_sptr self*, *bool en=True*)

`freq_sink_f_sptr.`**`enable_grid`**(*freq_sink_f_sptr self*, *bool en=True*)

`freq_sink_f_sptr.`**`enable_max_hold`**(*freq_sink_f_sptr self*, *bool en*)

`freq_sink_f_sptr.`**`enable_menu`**(*freq_sink_f_sptr self*, *bool en=True*)

`freq_sink_f_sptr.`**`enable_min_hold`**(*freq_sink_f_sptr self*, *bool en*)

`freq_sink_f_sptr.`**`exec_`**(*freq_sink_f_sptr self*)

`freq_sink_f_sptr.`**`fft_average`**(*freq_sink_f_sptr self*) → float

`freq_sink_f_sptr.`**`fft_size`**(*freq_sink_f_sptr self*) → int

`freq_sink_f_sptr.`**`fft_window`**(*freq_sink_f_sptr self*) → gr::filter::firdes::win_type

`freq_sink_f_sptr.`**`line_alpha`**(*freq_sink_f_sptr self*, *int which*) → double

`freq_sink_f_sptr.`**`line_color`**(*freq_sink_f_sptr self*, *int which*) → std::string

`freq_sink_f_sptr.`**`line_label`**(*freq_sink_f_sptr self*, *int which*) → std::string

`freq_sink_f_sptr.`**`line_marker`**(*freq_sink_f_sptr self*, *int which*) → int

`freq_sink_f_sptr.`**`line_style`**(*freq_sink_f_sptr self*, *int which*) → int

`freq_sink_f_sptr.`**`line_width`**(*freq_sink_f_sptr self*, *int which*) → int

`freq_sink_f_sptr.`**`message_subscribers`**(*freq_sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

`freq_sink_f_sptr.`**`min_noutput_items`**(*freq_sink_f_sptr self*) → int freq_sink_f_sptr.**pc_input_buffers_full_avg**(*freq_sink_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(freq_sink_f_sptr self) -> pmt_vector_float

freq_sink_f_sptr.**pc_noutput_items_avg**(*freq_sink_f_sptr self*) → float

freq_sink_f_sptr.**pc_nproduced_avg**(*freq_sink_f_sptr self*) → float

freq_sink_f_sptr.**pc_output_buffers_full_avg**(*freq_sink_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(freq_sink_f_sptr self) -> pmt_vector_float

freq_sink_f_sptr.**pc_throughput_avg**(*freq_sink_f_sptr self*) → float

freq_sink_f_sptr.**pc_work_time_avg**(*freq_sink_f_sptr self*) → float

freq_sink_f_sptr.**pc_work_time_total**(*freq_sink_f_sptr self*) → float

freq_sink_f_sptr.**pyqwidget**(*freq_sink_f_sptr self*) → PyObject *

freq_sink_f_sptr.**qwidget**(*freq_sink_f_sptr self*) → QWidget *

freq_sink_f_sptr.**reset**(*freq_sink_f_sptr self*)

freq_sink_f_sptr.**sample_delay**(*freq_sink_f_sptr self*, *int which*) → unsigned int

freq_sink_f_sptr.**set_fft_average**(*freq_sink_f_sptr self*, *float const fftavg*)

freq_sink_f_sptr.**set_fft_size**(*freq_sink_f_sptr self*, *int const fftsize*)

freq_sink_f_sptr.**set_fft_window**(*freq_sink_f_sptr self*, *gr::filter::firdes::win_type const win*)

freq_sink_f_sptr.**set_frequency_range**(*freq_sink_f_sptr self*, *double const centerfreq*, *double const bandwidth*)

freq_sink_f_sptr.**set_line_alpha**(*freq_sink_f_sptr self*, *int which*, *double alpha*)

freq_sink_f_sptr.**set_line_color**(*freq_sink_f_sptr self*, *int which*, *std::string const & color*)

freq_sink_f_sptr.**set_line_label**(*freq_sink_f_sptr self*, *int which*, *std::string const & label*)

freq_sink_f_sptr.**set_line_marker**(*freq_sink_f_sptr self*, *int which*, *int marker*)

freq_sink_f_sptr.**set_line_style**(*freq_sink_f_sptr self*, *int which*, *int style*)

freq_sink_f_sptr.**set_line_width**(*freq_sink_f_sptr self*, *int which*, *int width*)

freq_sink_f_sptr.**set_min_noutput_items**(*freq_sink_f_sptr self*, *int m*)

freq_sink_f_sptr.**set_plot_pos_half**(*freq_sink_f_sptr self*, *bool half*)

Pass "true" to this function to only show the positive half of the spectrum. By default, this plotter shows the full spectrum (positive and negative halves).

freq_sink_f_sptr.**set_size**(*freq_sink_f_sptr self*, *int width*, *int height*)

freq_sink_f_sptr.**set_thread_priority**(*freq_sink_f_sptr self*, *int priority*) → int

freq_sink_f_sptr.**set_title**(*freq_sink_f_sptr self*, *std::string const & title*)

freq_sink_f_sptr.**set_trigger_mode**(*freq_sink_f_sptr self*, *gr::qtgui::trigger_mode mode*, *float level*, *int channel*, *std::string const & tag_key*)

Set up a trigger for the sink to know when to start plotting. Useful to isolate events and avoid noise.

The trigger modes are Free, Auto, Normal, and Tag (see gr::qtgui::trigger_mode). The first three are like a normal trigger function. Free means free running with no trigger, auto will trigger if the trigger event is seen, but will still plot otherwise, and normal will hold until the trigger event is observed. The Tag trigger mode allows us to trigger off a specific stream tag. The tag trigger is based only on the name of the tag, so when a tag of the given name is seen, the trigger is activated.

In auto and normal mode, we look to see if the magnitude of the any FFT point is over the set level.

`freq_sink_f_sptr.`**`set_update_time`**(*freq_sink_f_sptr self*, *double t*)

`freq_sink_f_sptr.`**`set_y_axis`**(*freq_sink_f_sptr self*, *double min*, *double max*)

`freq_sink_f_sptr.`**`set_y_label`**(*freq_sink_f_sptr self*, *std::string const & label*, *std::string const & unit*)

`freq_sink_f_sptr.`**`thread_priority`**(*freq_sink_f_sptr self*) → int

`freq_sink_f_sptr.`**`title`**(*freq_sink_f_sptr self*) → std::string

`gnuradio.qtgui.`**`histogram_sink_f`**(*int size*, *int bins*, *double xmin*, *double xmax*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → histogram_sink_f_sptr

A graphical sink to display a histogram.

This is a QT-based graphical sink the displays a histogram of the data.

This histogram allows you to set and change at runtime the number of points to plot at once and the number of bins in the histogram. Both x and y-axis have their own auto-scaling behavior. By default, auto-scaling the y-axis is turned on and continuously updates the y-axis max value based on the currently plotted histogram.

The x-axis auto-scaling function only updates once when clicked. This resets the x-axis to the current range of minimum and maximum values represented in the histogram. It resets any values currently displayed because the location and width of the bins may have changed.

The histogram also has an accumulate function that simply accumulates the data between calls to work. When accumulate is activated, the y-axis autoscaling is turned on by default as the values will quickly grow in the this direction.

The sink supports plotting streaming float data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Float Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of float 32 values (pmt::is_f32vector) or PDUs where the data is a uniform vector of float 32 values.

Constructor Specific Documentation:

Build floating point histogram sink.

Parameters:

- **size** – number of points to plot at once
- **bins** – number of bins to sort the data into
- **xmin** – minimum x-axis value
- **xmax** – maximum x-axis value
- **name** – title for the plot
- **nconnections** – number of signals connected to sink
- **parent** – a QWidget parent object, if any

histogram_sink_f_sptr.**active_thread_priority**(*histogram_sink_f_sptr self*) → int

histogram_sink_f_sptr.**autoscalex**(*histogram_sink_f_sptr self*)

histogram_sink_f_sptr.**bins**(*histogram_sink_f_sptr self*) → int

histogram_sink_f_sptr.**d_qApplication**(*histogram_sink_f_sptr self*) → QApplication *

histogram_sink_f_sptr.**declare_sample_delay**(*histogram_sink_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(histogram_sink_f_sptr self, unsigned int delay)

histogram_sink_f_sptr.**disable_legend**(*histogram_sink_f_sptr self*)

histogram_sink_f_sptr.**enable_accumulate**(*histogram_sink_f_sptr self*, *bool en=True*)

histogram_sink_f_sptr.**enable_autoscale**(*histogram_sink_f_sptr self*, *bool en=True*)

histogram_sink_f_sptr.**enable_axis_labels**(*histogram_sink_f_sptr self*, *bool en=True*)

histogram_sink_f_sptr.**enable_grid**(*histogram_sink_f_sptr self*, *bool en=True*)

histogram_sink_f_sptr.**enable_menu**(*histogram_sink_f_sptr self*, *bool en=True*)

histogram_sink_f_sptr.**enable_semilogx**(*histogram_sink_f_sptr self*, *bool en=True*)

histogram_sink_f_sptr.**enable_semilogy**(*histogram_sink_f_sptr self*, *bool en=True*)

histogram_sink_f_sptr.**exec_**(*histogram_sink_f_sptr self*)

histogram_sink_f_sptr.**line_alpha**(*histogram_sink_f_sptr self*, *int which*) → double

histogram_sink_f_sptr.**line_color**(*histogram_sink_f_sptr self*, *int which*) → std::string

histogram_sink_f_sptr.**line_label**(*histogram_sink_f_sptr self*, *int which*) → std::string

histogram_sink_f_sptr.**line_marker**(*histogram_sink_f_sptr self*, *int which*) → int

histogram_sink_f_sptr.**line_style**(*histogram_sink_f_sptr self*, *int which*) → int

histogram_sink_f_sptr.**line_width**(*histogram_sink_f_sptr self*, *int which*) → int

histogram_sink_f_sptr.**message_subscribers**(*histogram_sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

histogram_sink_f_sptr.**min_noutput_items**(*histogram_sink_f_sptr self*) → int

histogram_sink_f_sptr.**nsamps**(*histogram_sink_f_sptr self*) → int

histogram_sink_f_sptr.**pc_input_buffers_full_avg**(*histogram_sink_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(histogram_sink_f_sptr self) -> pmt_vector_float

histogram_sink_f_sptr.**pc_noutput_items_avg**(*histogram_sink_f_sptr self*) → float `histogram_sink_f_sptr.`**`pc_nproduced_avg`**(*histogram_sink_f_sptr self*) → float

`histogram_sink_f_sptr.`**`pc_output_buffers_full_avg`**(*histogram_sink_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(histogram_sink_f_sptr self) -> pmt_vector_float

`histogram_sink_f_sptr.`**`pc_throughput_avg`**(*histogram_sink_f_sptr self*) → float

`histogram_sink_f_sptr.`**`pc_work_time_avg`**(*histogram_sink_f_sptr self*) → float

`histogram_sink_f_sptr.`**`pc_work_time_total`**(*histogram_sink_f_sptr self*) → float

`histogram_sink_f_sptr.`**`pyqwidget`**(*histogram_sink_f_sptr self*) → PyObject *

`histogram_sink_f_sptr.`**`qwidget`**(*histogram_sink_f_sptr self*) → QWidget *

`histogram_sink_f_sptr.`**`reset`**(*histogram_sink_f_sptr self*)

`histogram_sink_f_sptr.`**`sample_delay`**(*histogram_sink_f_sptr self*, *int which*) → unsigned int

`histogram_sink_f_sptr.`**`set_bins`**(*histogram_sink_f_sptr self*, *int const bins*)

`histogram_sink_f_sptr.`**`set_line_alpha`**(*histogram_sink_f_sptr self*, *int which*, *double alpha*)

`histogram_sink_f_sptr.`**`set_line_color`**(*histogram_sink_f_sptr self*, *int which*, *std::string const & color*)

`histogram_sink_f_sptr.`**`set_line_label`**(*histogram_sink_f_sptr self*, *int which*, *std::string const & line*)

`histogram_sink_f_sptr.`**`set_line_marker`**(*histogram_sink_f_sptr self*, *int which*, *int marker*)

`histogram_sink_f_sptr.`**`set_line_style`**(*histogram_sink_f_sptr self*, *int which*, *int style*)

`histogram_sink_f_sptr.`**`set_line_width`**(*histogram_sink_f_sptr self*, *int which*, *int width*)

`histogram_sink_f_sptr.`**`set_min_noutput_items`**(*histogram_sink_f_sptr self*, *int m*)

`histogram_sink_f_sptr.`**`set_nsamps`**(*histogram_sink_f_sptr self*, *int const newsize*)

`histogram_sink_f_sptr.`**`set_size`**(*histogram_sink_f_sptr self*, *int width*, *int height*)

`histogram_sink_f_sptr.`**`set_thread_priority`**(*histogram_sink_f_sptr self*, *int priority*) → int

`histogram_sink_f_sptr.`**`set_title`**(*histogram_sink_f_sptr self*, *std::string const & title*)

`histogram_sink_f_sptr.`**`set_update_time`**(*histogram_sink_f_sptr self*, *double t*)

`histogram_sink_f_sptr.`**`set_x_axis`**(*histogram_sink_f_sptr self*, *double min*, *double max*)

`histogram_sink_f_sptr.`**`set_y_axis`**(*histogram_sink_f_sptr self*, *double min*, *double max*) histogram_sink_f_sptr.**thread_priority**(*histogram_sink_f_sptr self*) → int

histogram_sink_f_sptr.**title**(*histogram_sink_f_sptr self*) → std::string

gnuradio.qtgui.**number_sink**(*size_t itemsize*, *float average=0*, *gr::qtgui::graph_t graph_type*, *int nconnections=1*, *QWidget * parent=None*) → number_sink_sptr

A graphical sink to display numerical values of input streams.

Displays the data stream in as a number in a simple text box GUI along with an optional bar graph. The bar graph can be set to horizontal (NUM_GRAPH_HORIZ), vertical (NUM_GRAPH_VERT), or no graph (NUM_GRAPH_NONE).

The displayed value can be the average of the input stream, in which case all items received are averaged. If not averaging, the display simply samples a value in the data stream based on the update time of this block.

Note that due to a flaw in the implementation, this block cannot receive integer value inputs. It will take chars, shorts, and floats and properly convert them by setting itemsize of the constructor to one of these three values (sizeof_char, sizeof_short, and sizeof_float, respectively). If using integers, the block treats these as floats. Instead, put the integer input stream through an gr::blocks::int_to_float converter block.

Constructor Specific Documentation:

Build a number sink.

Parameters:
- **itemsize** – Size of input item stream
- **average** – Averaging coefficient (0 - 1)
- **graph_type** – Type of graph to use (number_sink::graph_t)
- **nconnections** – number of signals connected to sink
- **parent** – a QWidget parent object, if any

number_sink_sptr.**active_thread_priority**(*number_sink_sptr self*) → int

number_sink_sptr.**average**(*number_sink_sptr self*) → float

number_sink_sptr.**color_max**(*number_sink_sptr self*, *int which*) → std::string

number_sink_sptr.**color_min**(*number_sink_sptr self*, *int which*) → std::string

number_sink_sptr.**d_qApplication**(*number_sink_sptr self*) → QApplication *

number_sink_sptr.**declare_sample_delay**(*number_sink_sptr self*, *int which*, *int delay*)

declare_sample_delay(number_sink_sptr self, unsigned int delay)

number_sink_sptr.**enable_autoscale**(*number_sink_sptr self*, *bool en=True*)

number_sink_sptr.**enable_menu**(*number_sink_sptr self*, *bool en=True*)

number_sink_sptr.**exec_**(*number_sink_sptr self*)

number_sink_sptr.**factor**(*number_sink_sptr self*, *int which*) → float

number_sink_sptr.**graph_type**(*number_sink_sptr self*) → gr::qtgui::graph_t

number_sink_sptr.**label**(*number_sink_sptr self*, *int which*) → std::string

number_sink_sptr.**max**(*number_sink_sptr self*, *int which*) → float

number_sink_sptr.**message_subscribers**(*number_sink_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

number_sink_sptr.**min**(*number_sink_sptr self*, *int which*) → float

number_sink_sptr.**min_noutput_items**(*number_sink_sptr self*) → int number_sink_sptr.**pc_input_buffers_full_avg**(*number_sink_sptr self*, *int which*) → float

pc_input_buffers_full_avg(number_sink_sptr self) -> pmt_vector_float

number_sink_sptr.**pc_noutput_items_avg**(*number_sink_sptr self*) → float

number_sink_sptr.**pc_nproduced_avg**(*number_sink_sptr self*) → float

number_sink_sptr.**pc_output_buffers_full_avg**(*number_sink_sptr self*, *int which*) → float

pc_output_buffers_full_avg(number_sink_sptr self) -> pmt_vector_float

number_sink_sptr.**pc_throughput_avg**(*number_sink_sptr self*) → float

number_sink_sptr.**pc_work_time_avg**(*number_sink_sptr self*) → float

number_sink_sptr.**pc_work_time_total**(*number_sink_sptr self*) → float

number_sink_sptr.**pyqwidget**(*number_sink_sptr self*) → PyObject *

number_sink_sptr.**qwidget**(*number_sink_sptr self*) → QWidget *

number_sink_sptr.**reset**(*number_sink_sptr self*)

number_sink_sptr.**sample_delay**(*number_sink_sptr self*, *int which*) → unsigned int

number_sink_sptr.**set_average**(*number_sink_sptr self*, *float const avg*)

number_sink_sptr.**set_color**(*number_sink_sptr self*, *int which*, *std::string const & min*, *std::string const & max*)

set_color(number_sink_sptr self, int which, int min, int max)

number_sink_sptr.**set_factor**(*number_sink_sptr self*, *int which*, *float factor*)

number_sink_sptr.**set_graph_type**(*number_sink_sptr self*, *gr::qtgui::graph_t const type*)

number_sink_sptr.**set_label**(*number_sink_sptr self*, *int which*, *std::string const & label*)

number_sink_sptr.**set_max**(*number_sink_sptr self*, *int which*, *float max*)

number_sink_sptr.**set_min**(*number_sink_sptr self*, *int which*, *float min*)

number_sink_sptr.**set_min_noutput_items**(*number_sink_sptr self*, *int m*)

number_sink_sptr.**set_thread_priority**(*number_sink_sptr self*, *int priority*) → int

number_sink_sptr.**set_title**(*number_sink_sptr self*, *std::string const & title*)

number_sink_sptr.**set_unit**(*number_sink_sptr self*, *int which*, *std::string const & unit*)

number_sink_sptr.**set_update_time**(*number_sink_sptr self*, *double t*)

number_sink_sptr.**thread_priority**(*number_sink_sptr self*) → int

number_sink_sptr.**title**(*number_sink_sptr self*) → std::string

number_sink_sptr.**unit**(*number_sink_sptr self*, *int which*) → std::string

gnuradio.qtgui.**sink_c**(*int fftsize*, *int wintype*, *double fc*, *double bw*, *std::string const & name*, *bool plotfreq*, *bool plotwaterfall*, *bool plottime*, *bool plotconst*, *QWidget ** *parent=None*) → sink_c_sptr

A graphical sink to display freq, spec, time, and const plots.

This is a QT-based graphical sink the takes a complex stream and plots it. The default action is to plot the signal as a PSD (FFT), spectrogram (waterfall), time domain I&Q, and constellation (I vs. Q) plots. The plots may be turned off by setting the appropriate boolean value in the constructor to False.

Message Ports:

Constructor Specific Documentation:

Build a complex qtgui sink.

Parameters:

- **fftsize** – size of the FFT to compute and display
- **wintype** – type of window to apply (see gnuradio/filter/firdes.h)
- **fc** – center frequency of signal (use for x-axis labels)
- **bw** – bandwidth of signal (used to set x-axis labels)
- **name** – title for the plot
- **plotfreq** – Toggle frequency plot on/off
- **plotwaterfall** – Toggle waterfall plot on/off
- **plottime** – Toggle time plot on/off
- **plotconst** – Toggle constellation plot on/off
- **parent** – a QWidget parent object, if any

`sink_c_sptr.`**`active_thread_priority`**(*sink_c_sptr self*) → int

`sink_c_sptr.`**`d_qApplication`**(*sink_c_sptr self*) → QApplication *

`sink_c_sptr.`**`declare_sample_delay`**(*sink_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(sink_c_sptr self, unsigned int delay)

`sink_c_sptr.`**`enable_rf_freq`**(*sink_c_sptr self*, *bool en*)

`sink_c_sptr.`**`exec_`**(*sink_c_sptr self*)

`sink_c_sptr.`**`fft_size`**(*sink_c_sptr self*) → int

`sink_c_sptr.`**`message_subscribers`**(*sink_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

`sink_c_sptr.`**`min_noutput_items`**(*sink_c_sptr self*) → int

`sink_c_sptr.`**`pc_input_buffers_full_avg`**(*sink_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sink_c_sptr self) -> pmt_vector_float

`sink_c_sptr.`**`pc_noutput_items_avg`**(*sink_c_sptr self*) → float

`sink_c_sptr.`**`pc_nproduced_avg`**(*sink_c_sptr self*) → float

`sink_c_sptr.`**`pc_output_buffers_full_avg`**(*sink_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sink_c_sptr self) -> pmt_vector_float

`sink_c_sptr.`**`pc_throughput_avg`**(*sink_c_sptr self*) → float

`sink_c_sptr.`**`pc_work_time_avg`**(*sink_c_sptr self*) → float

`sink_c_sptr.`**`pc_work_time_total`**(*sink_c_sptr self*) → float

`sink_c_sptr.`**`pyqwidget`**(*sink_c_sptr self*) → PyObject *

`sink_c_sptr.`**`qwidget`**(*sink_c_sptr self*) → QWidget *

`sink_c_sptr.`**`sample_delay`**(*sink_c_sptr self*, *int which*) → unsigned int sink_c_sptr.**set_fft_power_db**(*sink_c_sptr self*, *double min*, *double max*)

sink_c_sptr.**set_fft_size**(*sink_c_sptr self*, *int const fftsize*)

sink_c_sptr.**set_frequency_range**(*sink_c_sptr self*, *double const centerfreq*, *double const bandwidth*)

sink_c_sptr.**set_min_noutput_items**(*sink_c_sptr self*, *int m*)

sink_c_sptr.**set_thread_priority**(*sink_c_sptr self*, *int priority*) → int

sink_c_sptr.**set_update_time**(*sink_c_sptr self*, *double t*)

sink_c_sptr.**thread_priority**(*sink_c_sptr self*) → int

gnuradio.qtgui.**sink_f**(*int fftsize*, *int wintype*, *double fc*, *double bw*, *std::string const & name*, *bool plotfreq*, *bool plotwaterfall*, *bool plottime*, *bool plotconst*, *QWidget * parent=None*) → sink_f_sptr

A graphical sink to display freq, spec, and time.

This is a QT-based graphical sink the takes a float stream and plots it. The default action is to plot the signal as a PSD (FFT), spectrogram (waterfall), and time domain plots. The plots may be turned off by setting the appropriate boolean value in the constructor to False.

Message Ports:

Constructor Specific Documentation:

Build a floating point qtgui sink.

Parameters:
- **fftsize** – size of the FFT to compute and display
- **wintype** – type of window to apply (see gnuradio/filter/firdes.h)
- **fc** – center frequency of signal (use for x-axis labels)
- **bw** – bandwidth of signal (used to set x-axis labels)
- **name** – title for the plot
- **plotfreq** – Toggle frequency plot on/off
- **plotwaterfall** – Toggle waterfall plot on/off
- **plottime** – Toggle time plot on/off
- **plotconst** – Toggle constellation plot on/off
- **parent** – a QWidget parent object, if any

sink_f_sptr.**active_thread_priority**(*sink_f_sptr self*) → int

sink_f_sptr.**d_qApplication**(*sink_f_sptr self*) → QApplication *

sink_f_sptr.**declare_sample_delay**(*sink_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(sink_f_sptr self, unsigned int delay)

sink_f_sptr.**enable_rf_freq**(*sink_f_sptr self*, *bool en*)

sink_f_sptr.**exec_**(*sink_f_sptr self*)

sink_f_sptr.**fft_size**(*sink_f_sptr self*) → int

sink_f_sptr.**message_subscribers**(*sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sink_f_sptr.**min_noutput_items**(*sink_f_sptr self*) → int

sink_f_sptr.**pc_input_buffers_full_avg**(*sink_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sink_f_sptr self) -> pmt_vector_float

sink_f_sptr.**pc_noutput_items_avg**(*sink_f_sptr self*) → float sink_f_sptr.**pc_nproduced_avg**(*sink_f_sptr self*) → float

sink_f_sptr.**pc_output_buffers_full_avg**(*sink_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sink_f_sptr self) -> pmt_vector_float

sink_f_sptr.**pc_throughput_avg**(*sink_f_sptr self*) → float

sink_f_sptr.**pc_work_time_avg**(*sink_f_sptr self*) → float

sink_f_sptr.**pc_work_time_total**(*sink_f_sptr self*) → float

sink_f_sptr.**pyqwidget**(*sink_f_sptr self*) → PyObject *

sink_f_sptr.**qwidget**(*sink_f_sptr self*) → QWidget *

sink_f_sptr.**sample_delay**(*sink_f_sptr self*, *int which*) → unsigned int

sink_f_sptr.**set_fft_power_db**(*sink_f_sptr self*, *double min*, *double max*)

sink_f_sptr.**set_fft_size**(*sink_f_sptr self*, *int const fftsize*)

sink_f_sptr.**set_frequency_range**(*sink_f_sptr self*, *double const centerfreq*, *double const bandwidth*)

sink_f_sptr.**set_min_noutput_items**(*sink_f_sptr self*, *int m*)

sink_f_sptr.**set_thread_priority**(*sink_f_sptr self*, *int priority*) → int

sink_f_sptr.**set_update_time**(*sink_f_sptr self*, *double t*)

sink_f_sptr.**thread_priority**(*sink_f_sptr self*) → int

gnuradio.qtgui.**time_raster_sink_b**(*double samp_rate*, *double rows*, *double cols*, *pmt_vector_float mult*, *pmt_vector_float offset*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → time_raster_sink_b_sptr

A graphical sink to display multiple signals on a time_raster plot.

This is a QT-based graphical sink that takes in byte streams and plots a time_raster (spectrogram) plot.

Input stream: This expects a bit stream (0, 1 in the LSB of a byte). It will display packed bytes but the display will have to be autoscaled.

The sink supports plotting streaming byte/char data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Float Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of byte/char values (pmt::is_{u,s}32vector) or PDUs where the data is a uniform vector of byte/char values.

Constructor Specific Documentation:

Build a bit time raster sink.

Parameters:	
	• **samp_rate** – sample rate of signal • **cols** – number of cols to plot • **rows** – number of rows to plot • **mult** – vector of floats as a scaling multiplier for each input stream • **offset** – vector of floats as an offset for each input stream • **name** – title for the plot • **nconnections** – number of streams connected • **parent** – a QWidget parent object, if any

time_raster_sink_b_sptr.**active_thread_priority**(*time_raster_sink_b_sptr* *self*) → int

time_raster_sink_b_sptr.**color_map**(*time_raster_sink_b_sptr self*, *int which*) → int

time_raster_sink_b_sptr.**d_qApplication**(*time_raster_sink_b_sptr self*) → QApplication *

time_raster_sink_b_sptr.**declare_sample_delay**(*time_raster_sink_b_sptr self*, *int which*, *int delay*)

declare_sample_delay(time_raster_sink_b_sptr self, unsigned int delay)

time_raster_sink_b_sptr.**enable_autoscale**(*time_raster_sink_b_sptr self*, *bool en*)

time_raster_sink_b_sptr.**enable_axis_labels**(*time_raster_sink_b_sptr self*, *bool en=True*)

time_raster_sink_b_sptr.**enable_grid**(*time_raster_sink_b_sptr self*, *bool en*)

time_raster_sink_b_sptr.**enable_menu**(*time_raster_sink_b_sptr self*, *bool en*)

time_raster_sink_b_sptr.**exec_**(*time_raster_sink_b_sptr self*)

time_raster_sink_b_sptr.**line_alpha**(*time_raster_sink_b_sptr self*, *int which*) → double

time_raster_sink_b_sptr.**line_color**(*time_raster_sink_b_sptr self*, *int which*) → std::string

time_raster_sink_b_sptr.**line_label**(*time_raster_sink_b_sptr self*, *int which*) → std::string

time_raster_sink_b_sptr.**line_marker**(*time_raster_sink_b_sptr self*, *int which*) → int

time_raster_sink_b_sptr.**line_style**(*time_raster_sink_b_sptr self*, *int which*) → int

time_raster_sink_b_sptr.**line_width**(*time_raster_sink_b_sptr self*, *int which*) → int

time_raster_sink_b_sptr.**message_subscribers**(*time_raster_sink_b_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

time_raster_sink_b_sptr.**min_noutput_items**(*time_raster_sink_b_sptr self*) → int

time_raster_sink_b_sptr.**num_cols**(*time_raster_sink_b_sptr self*) → double

time_raster_sink_b_sptr.**num_rows**(*time_raster_sink_b_sptr self*) → double

time_raster_sink_b_sptr.**pc_input_buffers_full_avg**(*time_raster_sink_b_sptr self*, *int which*) → float

pc_input_buffers_full_avg(time_raster_sink_b_sptr self) -> pmt_vector_float

time_raster_sink_b_sptr.**pc_noutput_items_avg**(*time_raster_sink_b_sptr self*) → float

time_raster_sink_b_sptr.**pc_nproduced_avg**(*time_raster_sink_b_sptr self*) → float

time_raster_sink_b_sptr.**pc_output_buffers_full_avg**(*time_raster_sink_b_sptr self*, *int which*) → float pc_output_buffers_full_avg(time_raster_sink_b_sptr self) -> pmt_vector_float

time_raster_sink_b_sptr.**pc_throughput_avg**(*time_raster_sink_b_sptr self*) → float

time_raster_sink_b_sptr.**pc_work_time_avg**(*time_raster_sink_b_sptr self*) → float

time_raster_sink_b_sptr.**pc_work_time_total**(*time_raster_sink_b_sptr self*) → float

time_raster_sink_b_sptr.**pyqwidget**(*time_raster_sink_b_sptr self*) → PyObject *

time_raster_sink_b_sptr.**qwidget**(*time_raster_sink_b_sptr self*) → QWidget *

time_raster_sink_b_sptr.**reset**(*time_raster_sink_b_sptr self*)

time_raster_sink_b_sptr.**sample_delay**(*time_raster_sink_b_sptr self*, *int which*) → unsigned int

time_raster_sink_b_sptr.**set_color_map**(*time_raster_sink_b_sptr self*, *int which*, *int const color*)

time_raster_sink_b_sptr.**set_intensity_range**(*time_raster_sink_b_sptr self*, *float min*, *float max*)

time_raster_sink_b_sptr.**set_line_alpha**(*time_raster_sink_b_sptr self*, *int which*, *double alpha*)

time_raster_sink_b_sptr.**set_line_color**(*time_raster_sink_b_sptr self*, *int which*, *std::string const & color*)

time_raster_sink_b_sptr.**set_line_label**(*time_raster_sink_b_sptr self*, *int which*, *std::string const & lable*)

time_raster_sink_b_sptr.**set_line_marker**(*time_raster_sink_b_sptr self*, *int which*, *QwtSymbol::Style marker*)

time_raster_sink_b_sptr.**set_line_style**(*time_raster_sink_b_sptr self*, *int which*, *Qt::PenStyle style*)

time_raster_sink_b_sptr.**set_line_width**(*time_raster_sink_b_sptr self*, *int which*, *int width*)

time_raster_sink_b_sptr.**set_min_noutput_items**(*time_raster_sink_b_sptr self*, *int m*)

time_raster_sink_b_sptr.**set_multiplier**(*time_raster_sink_b_sptr self*, *pmt_vector_float mult*)

time_raster_sink_b_sptr.**set_num_cols**(*time_raster_sink_b_sptr self*, *double cols*)

time_raster_sink_b_sptr.**set_num_rows**(*time_raster_sink_b_sptr self*, *double rows*)

time_raster_sink_b_sptr.**set_offset**(*time_raster_sink_b_sptr self*, *pmt_vector_float offset*)

time_raster_sink_b_sptr.**set_samp_rate**(*time_raster_sink_b_sptr self*, *double const samp_rate*)

time_raster_sink_b_sptr.**set_size**(*time_raster_sink_b_sptr self*, *int width*, *int height*) `time_raster_sink_b_sptr.`**`set_thread_priority`**(*time_raster_sink_b_sptr self*, *int priority*) → int

`time_raster_sink_b_sptr.`**`set_title`**(*time_raster_sink_b_sptr self*, *std::string const & title*)

`time_raster_sink_b_sptr.`**`set_update_time`**(*time_raster_sink_b_sptr self*, *double t*)

`time_raster_sink_b_sptr.`**`thread_priority`**(*time_raster_sink_b_sptr self*) → int

`time_raster_sink_b_sptr.`**`title`**(*time_raster_sink_b_sptr self*) → std::string

`gnuradio.qtgui.`**`time_raster_sink_f`**(*double samp_rate*, *double rows*, *double cols*, *pmt_vector_float mult*, *pmt_vector_float offset*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → time_raster_sink_f_sptr

A graphical sink to display multiple signals on a time_raster plot.

This is a QT-based graphical sink that takes set of a floating point streams and plots a time_raster (spectrogram) plot.

The sink supports plotting streaming float data or messages. The message port is named "in". The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the "Float Message" type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of float 32 values (pmt::is_f32vector) or PDUs where the data is a uniform vector of float 32 values.

Constructor Specific Documentation:

Build a floating point time raster sink.

Parameters:

- **samp_rate** – sample rate of signal
- **cols** – number of cols to plot
- **rows** – number of rows to plot
- **mult** – vector of floats as a scaling multiplier for each input stream
- **offset** – vector of floats as an offset for each input stream
- **name** – title for the plot
- **nconnections** – number of streams connected
- **parent** – a QWidget parent object, if any

`time_raster_sink_f_sptr.`**`active_thread_priority`**(*time_raster_sink_f_sptr self*) → int

`time_raster_sink_f_sptr.`**`color_map`**(*time_raster_sink_f_sptr self*, *int which*) → int

`time_raster_sink_f_sptr.`**`d_qApplication`**(*time_raster_sink_f_sptr self*) → QApplication *

`time_raster_sink_f_sptr.`**`declare_sample_delay`**(*time_raster_sink_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(time_raster_sink_f_sptr self, unsigned int delay)

`time_raster_sink_f_sptr.`**`enable_autoscale`**(*time_raster_sink_f_sptr self*, *bool en*)

`time_raster_sink_f_sptr.`**`enable_axis_labels`**(*time_raster_sink_f_sptr self*, *bool en=True*)

`time_raster_sink_f_sptr.`**`enable_grid`**(*time_raster_sink_f_sptr self*, *bool en*)

`time_raster_sink_f_sptr.`**`enable_menu`**(*time_raster_sink_f_sptr self*, *bool en*)

`time_raster_sink_f_sptr.`**`exec_`**(*time_raster_sink_f_sptr self*) time_raster_sink_f_sptr.**line_alpha**(*time_raster_sink_f_sptr self*, *int which*) → double

time_raster_sink_f_sptr.**line_color**(*time_raster_sink_f_sptr self*, *int which*) → std::string

time_raster_sink_f_sptr.**line_label**(*time_raster_sink_f_sptr self*, *int which*) → std::string

time_raster_sink_f_sptr.**line_marker**(*time_raster_sink_f_sptr self*, *int which*) → int

time_raster_sink_f_sptr.**line_style**(*time_raster_sink_f_sptr self*, *int which*) → int

time_raster_sink_f_sptr.**line_width**(*time_raster_sink_f_sptr self*, *int which*) → int

time_raster_sink_f_sptr.**message_subscribers**(*time_raster_sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

time_raster_sink_f_sptr.**min_noutput_items**(*time_raster_sink_f_sptr self*) → int

time_raster_sink_f_sptr.**num_cols**(*time_raster_sink_f_sptr self*) → double

time_raster_sink_f_sptr.**num_rows**(*time_raster_sink_f_sptr self*) → double

time_raster_sink_f_sptr.**pc_input_buffers_full_avg**(*time_raster_sink_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(time_raster_sink_f_sptr self) -> pmt_vector_float

time_raster_sink_f_sptr.**pc_noutput_items_avg**(*time_raster_sink_f_sptr self*) → float

time_raster_sink_f_sptr.**pc_nproduced_avg**(*time_raster_sink_f_sptr self*) → float

time_raster_sink_f_sptr.**pc_output_buffers_full_avg**(*time_raster_sink_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(time_raster_sink_f_sptr self) -> pmt_vector_float

time_raster_sink_f_sptr.**pc_throughput_avg**(*time_raster_sink_f_sptr self*) → float

time_raster_sink_f_sptr.**pc_work_time_avg**(*time_raster_sink_f_sptr self*) → float

time_raster_sink_f_sptr.**pc_work_time_total**(*time_raster_sink_f_sptr self*) → float

time_raster_sink_f_sptr.**pyqwidget**(*time_raster_sink_f_sptr self*) → PyObject *

time_raster_sink_f_sptr.**qwidget**(*time_raster_sink_f_sptr self*) → QWidget *

time_raster_sink_f_sptr.**reset**(*time_raster_sink_f_sptr self*)

time_raster_sink_f_sptr.**sample_delay**(*time_raster_sink_f_sptr self*, *int which*) → unsigned int

time_raster_sink_f_sptr.**set_color_map**(*time_raster_sink_f_sptr self*, *int which*, *int const color*)

time_raster_sink_f_sptr.**set_intensity_range**(*time_raster_sink_f_sptr self*, *float min*, *float max*)

time_raster_sink_f_sptr.**set_line_alpha**(*time_raster_sink_f_sptr self*, *int which*, *double alpha*)

time_raster_sink_f_sptr.**set_line_color**(*time_raster_sink_f_sptr self*, *int which*, *std::string const & color*)

time_raster_sink_f_sptr.**set_line_label**(*time_raster_sink_f_sptr self*, *int which*, *std::string const & lable*)

time_raster_sink_f_sptr.**set_line_marker**(*time_raster_sink_f_sptr self*, *int which*, *QwtSymbol::Style marker*)

time_raster_sink_f_sptr.**set_line_style**(*time_raster_sink_f_sptr self*, *int which*, *Qt::PenStyle style*)

time_raster_sink_f_sptr.**set_line_width**(*time_raster_sink_f_sptr self*, *int which*, *int width*)

time_raster_sink_f_sptr.**set_min_noutput_items**(*time_raster_sink_f_sptr self*, *int m*)

time_raster_sink_f_sptr.**set_multiplier**(*time_raster_sink_f_sptr self*, *pmt_vector_float mult*)

time_raster_sink_f_sptr.**set_num_cols**(*time_raster_sink_f_sptr self*, *double cols*)

time_raster_sink_f_sptr.**set_num_rows**(*time_raster_sink_f_sptr self*, *double rows*)

time_raster_sink_f_sptr.**set_offset**(*time_raster_sink_f_sptr self*, *pmt_vector_float offset*)

time_raster_sink_f_sptr.**set_samp_rate**(*time_raster_sink_f_sptr self*, *double const samp_rate*)

time_raster_sink_f_sptr.**set_size**(*time_raster_sink_f_sptr self*, *int width*, *int height*)

time_raster_sink_f_sptr.**set_thread_priority**(*time_raster_sink_f_sptr self*, *int priority*) → int

time_raster_sink_f_sptr.**set_title**(*time_raster_sink_f_sptr self*, *std::string const & title*)

time_raster_sink_f_sptr.**set_update_time**(*time_raster_sink_f_sptr self*, *double t*)

time_raster_sink_f_sptr.**thread_priority**(*time_raster_sink_f_sptr self*) → int

time_raster_sink_f_sptr.**title**(*time_raster_sink_f_sptr self*) → std::string

gnuradio.qtgui.**time_sink_c**(*int size*, *double samp_rate*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → time_sink_c_sptr

A graphical sink to display multiple signals in time.

This is a QT-based graphical sink the takes set of a complex streams and plots them in the time domain. For each signal, both the signal's I and Q parts are plotted, and they are all plotted with a different color, and the and functions can be used to change the lable and color for a given input number.

The sink supports plotting streaming complex data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Complex Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of complex 32 values (pmt::is_c32vector) or PDUs where the data is a uniform vector of complex 32 values.

Constructor Specific Documentation:

Build complex time sink.

Parameters:

- **size** – number of points to plot at once
- **samp_rate** – sample rate (used to set x-axis labels)
- **name** – title for the plot
- **nconnections** – number of signals connected to sink
- **parent** – a QWidget parent object, if any

`time_sink_c_sptr.`**`active_thread_priority`**(*time_sink_c_sptr self*) → int

`time_sink_c_sptr.`**`d_qApplication`**(*time_sink_c_sptr self*) → QApplication *

`time_sink_c_sptr.`**`declare_sample_delay`**(*time_sink_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(time_sink_c_sptr self, unsigned int delay)

`time_sink_c_sptr.`**`disable_legend`**(*time_sink_c_sptr self*)

`time_sink_c_sptr.`**`enable_autoscale`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_axis_labels`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_control_panel`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_grid`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_menu`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_semilogx`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_semilogy`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_stem_plot`**(*time_sink_c_sptr self*, *bool en=True*)

`time_sink_c_sptr.`**`enable_tags`**(*time_sink_c_sptr self*, *int which*, *bool en*)

`time_sink_c_sptr.`**`exec_`**(*time_sink_c_sptr self*)

`time_sink_c_sptr.`**`line_alpha`**(*time_sink_c_sptr self*, *int which*) → double

`time_sink_c_sptr.`**`line_color`**(*time_sink_c_sptr self*, *int which*) → std::string

`time_sink_c_sptr.`**`line_label`**(*time_sink_c_sptr self*, *int which*) → std::string

`time_sink_c_sptr.`**`line_marker`**(*time_sink_c_sptr self*, *int which*) → int

`time_sink_c_sptr.`**`line_style`**(*time_sink_c_sptr self*, *int which*) → int

`time_sink_c_sptr.`**`line_width`**(*time_sink_c_sptr self*, *int which*) → int

`time_sink_c_sptr.`**`message_subscribers`**(*time_sink_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

`time_sink_c_sptr.`**`min_noutput_items`**(*time_sink_c_sptr self*) → int

`time_sink_c_sptr.`**`nsamps`**(*time_sink_c_sptr self*) → int time_sink_c_sptr.**pc_input_buffers_full_avg**(*time_sink_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(time_sink_c_sptr self) -> pmt_vector_float

time_sink_c_sptr.**pc_noutput_items_avg**(*time_sink_c_sptr self*) → float

time_sink_c_sptr.**pc_nproduced_avg**(*time_sink_c_sptr self*) → float

time_sink_c_sptr.**pc_output_buffers_full_avg**(*time_sink_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(time_sink_c_sptr self) -> pmt_vector_float

time_sink_c_sptr.**pc_throughput_avg**(*time_sink_c_sptr self*) → float

time_sink_c_sptr.**pc_work_time_avg**(*time_sink_c_sptr self*) → float

time_sink_c_sptr.**pc_work_time_total**(*time_sink_c_sptr self*) → float

time_sink_c_sptr.**pyqwidget**(*time_sink_c_sptr self*) → PyObject *

time_sink_c_sptr.**qwidget**(*time_sink_c_sptr self*) → QWidget *

time_sink_c_sptr.**reset**(*time_sink_c_sptr self*)

time_sink_c_sptr.**sample_delay**(*time_sink_c_sptr self*, *int which*) → unsigned int

time_sink_c_sptr.**set_line_alpha**(*time_sink_c_sptr self*, *int which*, *double alpha*)

time_sink_c_sptr.**set_line_color**(*time_sink_c_sptr self*, *int which*, *std::string const & color*)

time_sink_c_sptr.**set_line_label**(*time_sink_c_sptr self*, *int which*, *std::string const & label*)

time_sink_c_sptr.**set_line_marker**(*time_sink_c_sptr self*, *int which*, *int marker*)

time_sink_c_sptr.**set_line_style**(*time_sink_c_sptr self*, *int which*, *int style*)

time_sink_c_sptr.**set_line_width**(*time_sink_c_sptr self*, *int which*, *int width*)

time_sink_c_sptr.**set_min_noutput_items**(*time_sink_c_sptr self*, *int m*)

time_sink_c_sptr.**set_nsamps**(*time_sink_c_sptr self*, *int const newsize*)

time_sink_c_sptr.**set_samp_rate**(*time_sink_c_sptr self*, *double const samp_rate*)

time_sink_c_sptr.**set_size**(*time_sink_c_sptr self*, *int width*, *int height*)

time_sink_c_sptr.**set_thread_priority**(*time_sink_c_sptr self*, *int priority*) → int

time_sink_c_sptr.**set_title**(*time_sink_c_sptr self*, *std::string const & title*)

time_sink_c_sptr.**set_trigger_mode**(*time_sink_c_sptr self*, *gr::qtgui::trigger_mode mode*, *gr::qtgui::trigger_slope slope*, *float level*, *float delay*, *int channel*, *std::string const & tag_key*)

Set up a trigger for the sink to know when to start plotting. Useful to isolate events and avoid noise.

The trigger modes are Free, Auto, Normal, and Tag (see gr::qtgui::trigger_mode). The first three are like a normal oscope trigger function. Free means free running with no trigger, auto will trigger if the trigger event is seen, but will still plot otherwise, and normal will hold until the trigger event is observed. The Tag trigger mode allows us to trigger off a specific stream tag. The tag trigger is based only on the name of the tag, so when a tag of the given name is seen, the trigger is activated.

In auto and normal mode, we look for the slope of the of the signal. Given a gr::qtgui::trigger_slope as either Positive or Negative, if the value between two samples moves in the given direction (x[1] > x[0] for Positive or x[1] < x[0] for Negative), then the trigger is activated.

With the complex time sink, each input has two lines drawn for the real and imaginary parts of the signal. When selecting the value, channel 0 is the real signal and channel 1 is the imaginary signal. For more than 1 input stream, channel 2i is the real part of the ith input and channel (2i+1) is the imaginary part of the ith input channel.

The value is specified in time based off the sample rate. If the sample rate of the block is set to 1, the delay is then also the sample number offset. This is the offset from the left-hand y-axis of the plot. It delays the signal to show the trigger event at the given delay along with some portion of the signal before the event. The delay must be within 0 - t_max where t_max is the maximum amount of time displayed on the time plot.

`time_sink_c_sptr.`**`set_update_time`**(*time_sink_c_sptr self*, *double t*)

`time_sink_c_sptr.`**`set_y_axis`**(*time_sink_c_sptr self*, *double min*, *double max*)

`time_sink_c_sptr.`**`set_y_label`**(*time_sink_c_sptr self*, *std::string const & label*, *std::string const & unit*)

`time_sink_c_sptr.`**`thread_priority`**(*time_sink_c_sptr self*) → int

`time_sink_c_sptr.`**`title`**(*time_sink_c_sptr self*) → std::string

`gnuradio.qtgui.`**`time_sink_f`**(*int size*, *double samp_rate*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → time_sink_f_sptr

A graphical sink to display multiple signals in time.

This is a QT-based graphical sink the takes set of a float streams and plots them in the time domain. Each signal is plotted with a different color, and the and functions can be used to change the lable and color for a given input number.

The sink supports plotting streaming float data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Float Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of float 32 values (pmt::is_f32vector) or PDUs where the data is a uniform vector of float 32 values.

Constructor Specific Documentation:

Build floating point time sink.

Parameters:
- **size** – number of points to plot at once
- **samp_rate** – sample rate (used to set x-axis labels)
- **name** – title for the plot
- **nconnections** – number of signals connected to sink
- **parent** – a QWidget parent object, if any

`time_sink_f_sptr.`**`active_thread_priority`**(*time_sink_f_sptr self*) → int

`time_sink_f_sptr.`**`d_qApplication`**(*time_sink_f_sptr self*) → QApplication *

`time_sink_f_sptr.`**`declare_sample_delay`**(*time_sink_f_sptr self*, *int which*, *int delay*) declare_sample_delay(time_sink_f_sptr self, unsigned int delay)

time_sink_f_sptr.**disable_legend**(*time_sink_f_sptr self*)

time_sink_f_sptr.**enable_autoscale**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_axis_labels**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_control_panel**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_grid**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_menu**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_semilogx**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_semilogy**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_stem_plot**(*time_sink_f_sptr self*, *bool en=True*)

time_sink_f_sptr.**enable_tags**(*time_sink_f_sptr self*, *int which*, *bool en*)

time_sink_f_sptr.**exec_**(*time_sink_f_sptr self*)

time_sink_f_sptr.**line_alpha**(*time_sink_f_sptr self*, *int which*) → double

time_sink_f_sptr.**line_color**(*time_sink_f_sptr self*, *int which*) → std::string

time_sink_f_sptr.**line_label**(*time_sink_f_sptr self*, *int which*) → std::string

time_sink_f_sptr.**line_marker**(*time_sink_f_sptr self*, *int which*) → int

time_sink_f_sptr.**line_style**(*time_sink_f_sptr self*, *int which*) → int

time_sink_f_sptr.**line_width**(*time_sink_f_sptr self*, *int which*) → int

time_sink_f_sptr.**message_subscribers**(*time_sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

time_sink_f_sptr.**min_noutput_items**(*time_sink_f_sptr self*) → int

time_sink_f_sptr.**nsamps**(*time_sink_f_sptr self*) → int

time_sink_f_sptr.**pc_input_buffers_full_avg**(*time_sink_f_sptr self*, *int which*) → float
pc_input_buffers_full_avg(time_sink_f_sptr self) -> pmt_vector_float

time_sink_f_sptr.**pc_noutput_items_avg**(*time_sink_f_sptr self*) → float

time_sink_f_sptr.**pc_nproduced_avg**(*time_sink_f_sptr self*) → float

time_sink_f_sptr.**pc_output_buffers_full_avg**(*time_sink_f_sptr self*, *int which*) → float
pc_output_buffers_full_avg(time_sink_f_sptr self) -> pmt_vector_float

time_sink_f_sptr.**pc_throughput_avg**(*time_sink_f_sptr self*) → float

time_sink_f_sptr.**pc_work_time_avg**(*time_sink_f_sptr self*) → float

time_sink_f_sptr.**pc_work_time_total**(*time_sink_f_sptr self*) → float

time_sink_f_sptr.**pyqwidget**(*time_sink_f_sptr self*) → PyObject *

time_sink_f_sptr.**qwidget**(*time_sink_f_sptr self*) → QWidget * `time_sink_f_sptr.`**`reset`**(*time_sink_f_sptr self*)

`time_sink_f_sptr.`**`sample_delay`**(*time_sink_f_sptr self*, *int which*) → unsigned int

`time_sink_f_sptr.`**`set_line_alpha`**(*time_sink_f_sptr self*, *int which*, *double alpha*)

`time_sink_f_sptr.`**`set_line_color`**(*time_sink_f_sptr self*, *int which*, *std::string const & color*)

`time_sink_f_sptr.`**`set_line_label`**(*time_sink_f_sptr self*, *int which*, *std::string const & line*)

`time_sink_f_sptr.`**`set_line_marker`**(*time_sink_f_sptr self*, *int which*, *int marker*)

`time_sink_f_sptr.`**`set_line_style`**(*time_sink_f_sptr self*, *int which*, *int style*)

`time_sink_f_sptr.`**`set_line_width`**(*time_sink_f_sptr self*, *int which*, *int width*)

`time_sink_f_sptr.`**`set_min_noutput_items`**(*time_sink_f_sptr self*, *int m*)

`time_sink_f_sptr.`**`set_nsamps`**(*time_sink_f_sptr self*, *int const newsize*)

`time_sink_f_sptr.`**`set_samp_rate`**(*time_sink_f_sptr self*, *double const samp_rate*)

`time_sink_f_sptr.`**`set_size`**(*time_sink_f_sptr self*, *int width*, *int height*)

`time_sink_f_sptr.`**`set_thread_priority`**(*time_sink_f_sptr self*, *int priority*) → int

`time_sink_f_sptr.`**`set_title`**(*time_sink_f_sptr self*, *std::string const & title*)

`time_sink_f_sptr.`**`set_trigger_mode`**(*time_sink_f_sptr self*, *gr::qtgui::trigger_mode mode*, *gr::qtgui::trigger_slope slope*, *float level*, *float delay*, *int channel*, *std::string const & tag_key*)

Set up a trigger for the sink to know when to start plotting. Useful to isolate events and avoid noise.

The trigger modes are Free, Auto, Normal, and Tag (see gr::qtgui::trigger_mode). The first three are like a normal oscope trigger function. Free means free running with no trigger, auto will trigger if the trigger event is seen, but will still plot otherwise, and normal will hold until the trigger event is observed. The Tag trigger mode allows us to trigger off a specific stream tag. The tag trigger is based only on the name of the tag, so when a tag of the given name is seen, the trigger is activated.

In auto and normal mode, we look for the slope of the of the signal. Given a gr::qtgui::trigger_slope as either Positive or Negative, if the value between two samples moves in the given direction (x[1] > x[0] for Positive or x[1] < x[0] for Negative), then the trigger is activated.

The value is specified in time based off the sample rate. If the sample rate of the block is set to 1, the delay is then also the sample number offset. This is the offset from the left-hand y-axis of the plot. It delays the signal to show the trigger event at the given delay along with some portion of the signal before the event. The delay must be within 0 - t_max where t_max is the maximum amount of time displayed on the time plot.

`time_sink_f_sptr.`**`set_update_time`**(*time_sink_f_sptr self*, *double t*)

`time_sink_f_sptr.`**`set_y_axis`**(*time_sink_f_sptr self*, *double min*, *double max*)

`time_sink_f_sptr.`**`set_y_label`**(*time_sink_f_sptr self*, *std::string const & label*, *std::string const & unit*) time_sink_f_sptr.**thread_priority**(*time_sink_f_sptr self*) → int

time_sink_f_sptr.**title**(*time_sink_f_sptr self*) → std::string

gnuradio.qtgui.**vector_sink_f**(*int vlen*, *double x_start*, *double x_step*, *std::string const & x_axis_label*, *std::string const & y_axis_label*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → vector_sink_f_sptr

A graphical sink to display multiple vector-based signals.

This is a QT-based graphical sink that plots vectors of data as-is. Each signal is plotted with a different color, and the set_title() and set_color() functions can be used to change the label and color for a given input number.

To specify units for the x- and y-axes, use the set_x_axis_units() and set_y_axis_units() functions. This does not change the x- and y-labels, which are either specified during construction, or by calling the set_x_axis_label() and set_y_axis_label() methods.

Constructor Specific Documentation:

Build a vector plotting sink.

Parameters:
- **vlen** – Vector length at input. This cannot be changed during operations.
- **x_start** – The x-Axis value of the first vector element
- **x_step** – The step with which x-Axis values increment
- **x_axis_label** – The X-Axis label
- **y_axis_label** – The Y-Axis label
- **name** – title for the plot
- **nconnections** – number of signals connected to sink
- **parent** – a QWidget parent object, if any

vector_sink_f_sptr.**active_thread_priority**(*vector_sink_f_sptr self*) → int

vector_sink_f_sptr.**clear_max_hold**(*vector_sink_f_sptr self*)

vector_sink_f_sptr.**clear_min_hold**(*vector_sink_f_sptr self*)

vector_sink_f_sptr.**d_qApplication**(*vector_sink_f_sptr self*) → QApplication *

vector_sink_f_sptr.**declare_sample_delay**(*vector_sink_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_sink_f_sptr self, unsigned int delay)

vector_sink_f_sptr.**enable_autoscale**(*vector_sink_f_sptr self*, *bool en=True*)

vector_sink_f_sptr.**enable_grid**(*vector_sink_f_sptr self*, *bool en=True*)

vector_sink_f_sptr.**enable_menu**(*vector_sink_f_sptr self*, *bool en=True*)

vector_sink_f_sptr.**exec_**(*vector_sink_f_sptr self*)

vector_sink_f_sptr.**line_alpha**(*vector_sink_f_sptr self*, *int which*) → double

vector_sink_f_sptr.**line_color**(*vector_sink_f_sptr self*, *int which*) → std::string

vector_sink_f_sptr.**line_label**(*vector_sink_f_sptr self*, *int which*) → std::string

vector_sink_f_sptr.**line_marker**(*vector_sink_f_sptr self*, *int which*) → int

vector_sink_f_sptr.**line_style**(*vector_sink_f_sptr self*, *int which*) → int

vector_sink_f_sptr.**line_width**(*vector_sink_f_sptr self*, *int which*) → int

vector_sink_f_sptr.**message_subscribers**(*vector_sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr `vector_sink_f_sptr.`**`min_noutput_items`**(*vector_sink_f_sptr self*) → int

`vector_sink_f_sptr.`**`pc_input_buffers_full_avg`**(*vector_sink_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_sink_f_sptr self) -> pmt_vector_float

`vector_sink_f_sptr.`**`pc_noutput_items_avg`**(*vector_sink_f_sptr self*) → float

`vector_sink_f_sptr.`**`pc_nproduced_avg`**(*vector_sink_f_sptr self*) → float

`vector_sink_f_sptr.`**`pc_output_buffers_full_avg`**(*vector_sink_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_sink_f_sptr self) -> pmt_vector_float

`vector_sink_f_sptr.`**`pc_throughput_avg`**(*vector_sink_f_sptr self*) → float

`vector_sink_f_sptr.`**`pc_work_time_avg`**(*vector_sink_f_sptr self*) → float

`vector_sink_f_sptr.`**`pc_work_time_total`**(*vector_sink_f_sptr self*) → float

`vector_sink_f_sptr.`**`pyqwidget`**(*vector_sink_f_sptr self*) → PyObject *

`vector_sink_f_sptr.`**`qwidget`**(*vector_sink_f_sptr self*) → QWidget *

`vector_sink_f_sptr.`**`reset`**(*vector_sink_f_sptr self*)

`vector_sink_f_sptr.`**`sample_delay`**(*vector_sink_f_sptr self*, *int which*) → unsigned int

`vector_sink_f_sptr.`**`set_line_alpha`**(*vector_sink_f_sptr self*, *int which*, *double alpha*)

`vector_sink_f_sptr.`**`set_line_color`**(*vector_sink_f_sptr self*, *int which*, *std::string const & color*)

`vector_sink_f_sptr.`**`set_line_label`**(*vector_sink_f_sptr self*, *int which*, *std::string const & label*)

`vector_sink_f_sptr.`**`set_line_marker`**(*vector_sink_f_sptr self*, *int which*, *int marker*)

`vector_sink_f_sptr.`**`set_line_style`**(*vector_sink_f_sptr self*, *int which*, *int style*)

`vector_sink_f_sptr.`**`set_line_width`**(*vector_sink_f_sptr self*, *int which*, *int width*)

`vector_sink_f_sptr.`**`set_min_noutput_items`**(*vector_sink_f_sptr self*, *int m*)

`vector_sink_f_sptr.`**`set_ref_level`**(*vector_sink_f_sptr self*, *double ref_level*)

The ref level is a reference line.

`vector_sink_f_sptr.`**`set_size`**(*vector_sink_f_sptr self*, *int width*, *int height*)

`vector_sink_f_sptr.`**`set_thread_priority`**(*vector_sink_f_sptr self*, *int priority*) → int

`vector_sink_f_sptr.`**`set_title`**(*vector_sink_f_sptr self*, *std::string const & title*)

`vector_sink_f_sptr.`**`set_update_time`**(*vector_sink_f_sptr self*, *double t*)

`vector_sink_f_sptr.`**`set_vec_average`**(*vector_sink_f_sptr self*, *float const avg*)

`vector_sink_f_sptr.`**`set_x_axis`**(*vector_sink_f_sptr self*, *double const start*, *double const step*)

Update the values on the x-Axis.

vector_sink_f_sptr.**set_x_axis_label**(*vector_sink_f_sptr self*, *std::string const & label*)

vector_sink_f_sptr.**set_x_axis_units**(*vector_sink_f_sptr self*, *std::string const & units*)

Change the units string on the x-Axis (e.g. ‘m’ if x-Axis label was ‘Distance’)

vector_sink_f_sptr.**set_y_axis**(*vector_sink_f_sptr self*, *double min*, *double max*)

vector_sink_f_sptr.**set_y_axis_label**(*vector_sink_f_sptr self*, *std::string const & label*)

vector_sink_f_sptr.**set_y_axis_units**(*vector_sink_f_sptr self*, *std::string const & units*)

Change the units string on the y-Axis (e.g. ‘V’ if x-Axis label was ‘Amplitude’)

vector_sink_f_sptr.**thread_priority**(*vector_sink_f_sptr self*) → int

vector_sink_f_sptr.**title**(*vector_sink_f_sptr self*) → std::string

vector_sink_f_sptr.**vec_average**(*vector_sink_f_sptr self*) → float

vector_sink_f_sptr.**vlen**(*vector_sink_f_sptr self*) → int

gnuradio.qtgui.**waterfall_sink_c**(*int size*, *int wintype*, *double fc*, *double bw*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → waterfall_sink_c_sptr

A graphical sink to display multiple signals on a waterfall (spectrogram) plot.

This is a QT-based graphical sink the takes set of a complex streams and plots a waterfall (spectrogram) plot.

Note that unlike the other qtgui sinks, this one does not support multiple input streams. We have yet to figure out a good way to display multiple, independent signals on this kind of a plot. If there are any suggestions or examples of this, we would love to see them. Otherwise, to display multiple signals here, it’s probably best to sum the signals together and connect that here.

The sink supports plotting streaming complex data or messages. The message port is named “in”. The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the “Complex Message” type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of complex 32 values (pmt::is_c32vector) or PDUs where the data is a uniform vector of complex 32 values.

Message Ports:

Constructor Specific Documentation:

Build a complex waterfall sink.

Parameters:

- **size** – size of the FFT to compute and display. If using the PDU message port to plot samples, the length of each PDU must be a multiple of the FFT size.
- **wintype** – type of window to apply (see gr::fft::window::win_type)
- **fc** – center frequency of signal (use for x-axis labels)
- **bw** – bandwidth of signal (used to set x-axis labels)
- **name** – title for the plot
- **nconnections** – number of signals to be connected to the sink. The PDU message port is always available for a connection, and this value must be set to 0 if only the PDU message port is being used.
- **parent** – a QWidget parent object, if any

`waterfall_sink_c_sptr.`**`active_thread_priority`**(*waterfall_sink_c_sptr self*) → int

`waterfall_sink_c_sptr.`**`auto_scale`**(*waterfall_sink_c_sptr self*)

`waterfall_sink_c_sptr.`**`clear_data`**(*waterfall_sink_c_sptr self*)

`waterfall_sink_c_sptr.`**`color_map`**(*waterfall_sink_c_sptr self*, *int which*) → int

`waterfall_sink_c_sptr.`**`d_qApplication`**(*waterfall_sink_c_sptr self*) → QApplication *

`waterfall_sink_c_sptr.`**`declare_sample_delay`**(*waterfall_sink_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(waterfall_sink_c_sptr self, unsigned int delay)

`waterfall_sink_c_sptr.`**`disable_legend`**(*waterfall_sink_c_sptr self*)

`waterfall_sink_c_sptr.`**`enable_axis_labels`**(*waterfall_sink_c_sptr self*, *bool en=True*)

`waterfall_sink_c_sptr.`**`enable_grid`**(*waterfall_sink_c_sptr self*, *bool en=True*)

`waterfall_sink_c_sptr.`**`enable_menu`**(*waterfall_sink_c_sptr self*, *bool en=True*)

`waterfall_sink_c_sptr.`**`exec_`**(*waterfall_sink_c_sptr self*)

`waterfall_sink_c_sptr.`**`fft_average`**(*waterfall_sink_c_sptr self*) → float

`waterfall_sink_c_sptr.`**`fft_size`**(*waterfall_sink_c_sptr self*) → int

`waterfall_sink_c_sptr.`**`fft_window`**(*waterfall_sink_c_sptr self*) → gr::filter::firdes::win_type

`waterfall_sink_c_sptr.`**`line_alpha`**(*waterfall_sink_c_sptr self*, *int which*) → double

`waterfall_sink_c_sptr.`**`line_label`**(*waterfall_sink_c_sptr self*, *int which*) → std::string

`waterfall_sink_c_sptr.`**`max_intensity`**(*waterfall_sink_c_sptr self*, *int which*) → double

`waterfall_sink_c_sptr.`**`message_subscribers`**(*waterfall_sink_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

`waterfall_sink_c_sptr.`**`min_intensity`**(*waterfall_sink_c_sptr self*, *int which*) → double

`waterfall_sink_c_sptr.`**`min_noutput_items`**(*waterfall_sink_c_sptr self*) → int

`waterfall_sink_c_sptr.`**`pc_input_buffers_full_avg`**(*waterfall_sink_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(waterfall_sink_c_sptr self) -> pmt_vector_float

`waterfall_sink_c_sptr.`**`pc_noutput_items_avg`**(*waterfall_sink_c_sptr self*) → float

`waterfall_sink_c_sptr.`**`pc_nproduced_avg`**(*waterfall_sink_c_sptr self*) → float

`waterfall_sink_c_sptr.`**`pc_output_buffers_full_avg`**(*waterfall_sink_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(waterfall_sink_c_sptr self) -> pmt_vector_float

`waterfall_sink_c_sptr.`**`pc_throughput_avg`**(*waterfall_sink_c_sptr self*) → float waterfall_sink_c_sptr.**pc_work_time_avg**(*waterfall_sink_c_sptr self*) → float

waterfall_sink_c_sptr.**pc_work_time_total**(*waterfall_sink_c_sptr self*) → float

waterfall_sink_c_sptr.**pyqwidget**(*waterfall_sink_c_sptr self*) → PyObject *

waterfall_sink_c_sptr.**qwidget**(*waterfall_sink_c_sptr self*) → QWidget *

waterfall_sink_c_sptr.**sample_delay**(*waterfall_sink_c_sptr self*, *int which*) → unsigned int

waterfall_sink_c_sptr.**set_color_map**(*waterfall_sink_c_sptr self*, *int which*, *int const color*)

waterfall_sink_c_sptr.**set_fft_average**(*waterfall_sink_c_sptr self*, *float const fftavg*)

waterfall_sink_c_sptr.**set_fft_size**(*waterfall_sink_c_sptr self*, *int const fftsize*)

waterfall_sink_c_sptr.**set_fft_window**(*waterfall_sink_c_sptr self*, *gr::filter::firdes::win_type const win*)

waterfall_sink_c_sptr.**set_frequency_range**(*waterfall_sink_c_sptr self*, *double const centerfreq*, *double const bandwidth*)

waterfall_sink_c_sptr.**set_intensity_range**(*waterfall_sink_c_sptr self*, *double const min*, *double const max*)

waterfall_sink_c_sptr.**set_line_alpha**(*waterfall_sink_c_sptr self*, *int which*, *double alpha*)

waterfall_sink_c_sptr.**set_line_label**(*waterfall_sink_c_sptr self*, *int which*, *std::string const & line*)

waterfall_sink_c_sptr.**set_min_noutput_items**(*waterfall_sink_c_sptr self*, *int m*)

waterfall_sink_c_sptr.**set_size**(*waterfall_sink_c_sptr self*, *int width*, *int height*)

waterfall_sink_c_sptr.**set_thread_priority**(*waterfall_sink_c_sptr self*, *int priority*) → int

waterfall_sink_c_sptr.**set_time_per_fft**(*waterfall_sink_c_sptr self*, *double const t*)

waterfall_sink_c_sptr.**set_time_title**(*waterfall_sink_c_sptr self*, *std::string const & title*)

waterfall_sink_c_sptr.**set_title**(*waterfall_sink_c_sptr self*, *std::string const & title*)

waterfall_sink_c_sptr.**set_update_time**(*waterfall_sink_c_sptr self*, *double t*)

waterfall_sink_c_sptr.**thread_priority**(*waterfall_sink_c_sptr self*) → int

waterfall_sink_c_sptr.**title**(*waterfall_sink_c_sptr self*) → std::string

gnuradio.qtgui.**waterfall_sink_f**(*int size*, *int wintype*, *double fc*, *double bw*, *std::string const & name*, *int nconnections=1*, *QWidget * parent=None*) → waterfall_sink_f_sptr

A graphical sink to display multiple signals on a waterfall (spectrogram) plot.

This is a QT-based graphical sink the takes set of a floating point streams and plots a waterfall (spectrogram) plot.

Note that unlike the other qtgui sinks, this one does not support multiple input streams. We have yet to figure out a good way to display multiple, independent signals on this kind of a plot. If there are any suggestions or examples of this, we would love to see them. Otherwise, to display multiple signals here, it's probably best to sum the signals together and connect that here.

The sink supports plotting streaming float data or messages. The message port is named "in". The two modes cannot be used simultaneously, and should be set to 0 when using the message mode. GRC handles this issue by providing the "Float Message" type that removes the streaming port(s).

This sink can plot messages that contain either uniform vectors of float 32 values (pmt::is_f32vector) or PDUs where the data is a uniform vector of float 32 values.

Message Ports:

Constructor Specific Documentation:

Build a floating point waterfall sink.

Parameters:
- **size** – size of the FFT to compute and display. If using the PDU message port to plot samples, the length of each PDU must be a multiple of the FFT size.
- **wintype** – type of window to apply (see gr::fft::window::win_type)
- **fc** – center frequency of signal (use for x-axis labels)
- **bw** – bandwidth of signal (used to set x-axis labels)
- **name** – title for the plot
- **nconnections** – number of signals to be connected to the sink. The PDU message port is always available for a connection, and this value must be set to 0 if only the PDU message port is being used.
- **parent** – a QWidget parent object, if any

`waterfall_sink_f_sptr.`**`active_thread_priority`**(*waterfall_sink_f_sptr self*) → int

`waterfall_sink_f_sptr.`**`auto_scale`**(*waterfall_sink_f_sptr self*)

`waterfall_sink_f_sptr.`**`clear_data`**(*waterfall_sink_f_sptr self*)

`waterfall_sink_f_sptr.`**`color_map`**(*waterfall_sink_f_sptr self*, *int which*) → int

`waterfall_sink_f_sptr.`**`d_qApplication`**(*waterfall_sink_f_sptr self*) → QApplication *

`waterfall_sink_f_sptr.`**`declare_sample_delay`**(*waterfall_sink_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(waterfall_sink_f_sptr self, unsigned int delay)

`waterfall_sink_f_sptr.`**`disable_legend`**(*waterfall_sink_f_sptr self*)

`waterfall_sink_f_sptr.`**`enable_axis_labels`**(*waterfall_sink_f_sptr self*, *bool en=True*)

`waterfall_sink_f_sptr.`**`enable_grid`**(*waterfall_sink_f_sptr self*, *bool en=True*)

`waterfall_sink_f_sptr.`**`enable_menu`**(*waterfall_sink_f_sptr self*, *bool en=True*)

`waterfall_sink_f_sptr.`**`exec_`**(*waterfall_sink_f_sptr self*)

`waterfall_sink_f_sptr.`**`fft_average`**(*waterfall_sink_f_sptr self*) → float

`waterfall_sink_f_sptr.`**`fft_size`**(*waterfall_sink_f_sptr self*) → int

`waterfall_sink_f_sptr.`**`fft_window`**(*waterfall_sink_f_sptr self*) → gr::filter::firdes::win_type `waterfall_sink_f_sptr.`**`line_alpha`**(*waterfall_sink_f_sptr self*, *int which*) → double

`waterfall_sink_f_sptr.`**`line_label`**(*waterfall_sink_f_sptr self*, *int which*) → std::string

`waterfall_sink_f_sptr.`**`max_intensity`**(*waterfall_sink_f_sptr self*, *int which*) → double

`waterfall_sink_f_sptr.`**`message_subscribers`**(*waterfall_sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

`waterfall_sink_f_sptr.`**`min_intensity`**(*waterfall_sink_f_sptr self*, *int which*) → double

`waterfall_sink_f_sptr.`**`min_noutput_items`**(*waterfall_sink_f_sptr self*) → int

`waterfall_sink_f_sptr.`**`pc_input_buffers_full_avg`**(*waterfall_sink_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(waterfall_sink_f_sptr self) -> pmt_vector_float

`waterfall_sink_f_sptr.`**`pc_noutput_items_avg`**(*waterfall_sink_f_sptr self*) → float

`waterfall_sink_f_sptr.`**`pc_nproduced_avg`**(*waterfall_sink_f_sptr self*) → float

`waterfall_sink_f_sptr.`**`pc_output_buffers_full_avg`**(*waterfall_sink_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(waterfall_sink_f_sptr self) -> pmt_vector_float

`waterfall_sink_f_sptr.`**`pc_throughput_avg`**(*waterfall_sink_f_sptr self*) → float

`waterfall_sink_f_sptr.`**`pc_work_time_avg`**(*waterfall_sink_f_sptr self*) → float

`waterfall_sink_f_sptr.`**`pc_work_time_total`**(*waterfall_sink_f_sptr self*) → float

`waterfall_sink_f_sptr.`**`pyqwidget`**(*waterfall_sink_f_sptr self*) → PyObject *

`waterfall_sink_f_sptr.`**`qwidget`**(*waterfall_sink_f_sptr self*) → QWidget *

`waterfall_sink_f_sptr.`**`sample_delay`**(*waterfall_sink_f_sptr self*, *int which*) → unsigned int

`waterfall_sink_f_sptr.`**`set_color_map`**(*waterfall_sink_f_sptr self*, *int which*, *int const color*)

`waterfall_sink_f_sptr.`**`set_fft_average`**(*waterfall_sink_f_sptr self*, *float const fftavg*)

`waterfall_sink_f_sptr.`**`set_fft_size`**(*waterfall_sink_f_sptr self*, *int const fftsize*)

`waterfall_sink_f_sptr.`**`set_fft_window`**(*waterfall_sink_f_sptr self*, *gr::filter::firdes::win_type const win*)

`waterfall_sink_f_sptr.`**`set_frequency_range`**(*waterfall_sink_f_sptr self*, *double const centerfreq*, *double const bandwidth*)

`waterfall_sink_f_sptr.`**`set_intensity_range`**(*waterfall_sink_f_sptr self*, *double const min*, *double const max*)

`waterfall_sink_f_sptr.`**`set_line_alpha`**(*waterfall_sink_f_sptr self*, *int which*, *double alpha*)

`waterfall_sink_f_sptr.`**`set_line_label`**(*waterfall_sink_f_sptr self*, *int which*, *std::string const & line*) waterfall_sink_f_sptr.**set_min_noutput_items**(*waterfall_sink_f_sptr self*, *int m*)

waterfall_sink_f_sptr.**set_plot_pos_half**(*waterfall_sink_f_sptr self*, *bool half*)

Pass “true” to this function to only show the positive half of the spectrum. By default, this plotter shows the full spectrum (positive and negative halves).

waterfall_sink_f_sptr.**set_size**(*waterfall_sink_f_sptr self*, *int width*, *int height*)

waterfall_sink_f_sptr.**set_thread_priority**(*waterfall_sink_f_sptr self*, *int priority*) → int

waterfall_sink_f_sptr.**set_time_per_fft**(*waterfall_sink_f_sptr self*, *double const t*)

waterfall_sink_f_sptr.**set_time_title**(*waterfall_sink_f_sptr self*, *std::string const & title*)

waterfall_sink_f_sptr.**set_title**(*waterfall_sink_f_sptr self*, *std::string const & title*)

waterfall_sink_f_sptr.**set_update_time**(*waterfall_sink_f_sptr self*, *double t*)

waterfall_sink_f_sptr.**thread_priority**(*waterfall_sink_f_sptr self*) → int

waterfall_sink_f_sptr.**title**(*waterfall_sink_f_sptr self*) → std::string