Previous topic
gnuradio.analog
Next topic
gnuradio.channels
This Page
Show Source
Quick search
Go
Enter search terms or a module, class or function name.

gnuradio.blocks

Processing blocks common to many flowgraphs.

gnuradio.blocks.**abs_ff**(*size_t vlen=1*) → abs_ff_sptr

output[m] = abs(input[m]) for all M streams.

absolute value of data stream (Strip sign)

Constructor Specific Documentation:

Create an instance of abs_ff.

Parameters: **vlen** –

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

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

declare_sample_delay(abs_ff_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(abs_ff_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(abs_ff_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**abs_ii**(*size_t vlen=1*) → abs_ii_sptr

output[m] = abs(input[m]) for all M streams.

absolute value of data stream (Strip sign)

Constructor Specific Documentation:

Create an instance of abs_ii.

Parameters: **vlen** –

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

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

declare_sample_delay(abs_ii_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(abs_ii_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**abs_ss**(*size_t vlen=1*) → abs_ss_sptr
output[m] = abs(input[m]) for all M streams.

absolute value of data stream (Strip sign)

Constructor Specific Documentation:

Create an instance of abs_ss.

Parameters: **vlen** –

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**add_cc**(*size_t vlen=1*) → add_cc_sptr
output = sum(input[0], input[1], ..., input[M-1])

Add samples across all input streams. For all samples on all input streams :

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**add_const_bb**(*unsigned char k*) → add_const_bb_sptr
output = input + constant

Constructor Specific Documentation:

Create an instance of add_const_bb.

Parameters: **k** – additive constant

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

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

add_const_bb_sptr.**k**(*add_const_bb_sptr self*) → unsigned char
Return additive constant.

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

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

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

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

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

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

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

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

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

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

add_const_bb_sptr.**set_k**(*add_const_bb_sptr self*, *unsigned char k*)
Set additive constant.

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

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

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

gnuradio.blocks.**add_const_cc**(*gr_complex k*) → add_const_cc_sptr
output = input + constant

Constructor Specific Documentation:

Create an instance of add_const_cc.

Parameters: **k** – additive constant

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

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

add_const_cc_sptr.**k**(*add_const_cc_sptr self*) → gr_complex
Return additive constant.

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

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

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

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

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

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

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

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

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

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

add_const_cc_sptr.**set_k**(*add_const_cc_sptr self*, *gr_complex k*)
Set additive constant.

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

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

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

gnuradio.blocks.**add_const_ff**(*float k*) → add_const_ff_sptr
output = input + constant

Constructor Specific Documentation:

Create an instance of add_const_ff.

Parameters: **k** – additive constant

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

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

add_const_ff_sptr.**k**(*add_const_ff_sptr self*) → float
Return additive constant.

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

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

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

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

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

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

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

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

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

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

add_const_ff_sptr.**set_k**(*add_const_ff_sptr self*, *float k*)

Set additive constant.

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

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

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

gnuradio.blocks.**add_const_ii**(*int k*) → add_const_ii_sptr

output = input + constant

Constructor Specific Documentation:

Create an instance of add_const_ii.

Parameters: **k** – additive constant

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

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

declare_sample_delay(add_const_ii_sptr self, unsigned int delay)

add_const_ii_sptr.**k**(*add_const_ii_sptr self*) → int

Return additive constant.

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

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

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

pc_input_buffers_full_avg(add_const_ii_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(add_const_ii_sptr self) -> pmt_vector_float

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

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

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

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

add_const_ii_sptr.**set_k**(*add_const_ii_sptr self*, *int k*)

Set additive constant.

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

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

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

gnuradio.blocks.**add_const_ss**(*short k*) → add_const_ss_sptr

output = input + constant

Constructor Specific Documentation:

Create an instance of add_const_ss.

Parameters: **k** – additive constant

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

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

declare_sample_delay(add_const_ss_sptr self, unsigned int delay)

add_const_ss_sptr.**k**(*add_const_ss_sptr self*) → short

Return additive constant.

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

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

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

pc_input_buffers_full_avg(add_const_ss_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(add_const_ss_sptr self) -> pmt_vector_float

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

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

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

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

add_const_ss_sptr.**set_k**(*add_const_ss_sptr self*, *short k*)

Set additive constant.

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

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

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

gnuradio.blocks.**add_const_vbb**(*std::vector< unsigned char, std::allocator< unsigned char > > k*) → add_const_vbb_sptr

output[m] = input[m] + constant vector for all M streams.

Constructor Specific Documentation:

Create an instance of add_const_vbb.

Parameters: **k** – additive constant vector

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

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

declare_sample_delay(add_const_vbb_sptr self, unsigned int delay)

add_const_vbb_sptr.**k**(*add_const_vbb_sptr self*) → std::vector< unsigned char,std::allocator< unsigned char > >

Return additive constant vector.

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

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

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

pc_input_buffers_full_avg(add_const_vbb_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(add_const_vbb_sptr self) -> pmt_vector_float

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

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

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

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

add_const_vbb_sptr.**set_k**(*add_const_vbb_sptr self*, *std::vector< unsigned char, std::allocator< unsigned char > > k*)

Set additive constant vector.

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

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

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

gnuradio.blocks.**add_const_vcc**(*pmt_vector_cfloat k*) → add_const_vcc_sptr

output[m] = input[m] + constant vector for all M streams.

Constructor Specific Documentation:

Create an instance of add_const_vcc.

Parameters: **k** – additive constant vector

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

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

declare_sample_delay(add_const_vcc_sptr self, unsigned int delay)

add_const_vcc_sptr.**k**(*add_const_vcc_sptr self*) → pmt_vector_cfloat

Return additive constant vector.

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

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

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

pc_input_buffers_full_avg(add_const_vcc_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(add_const_vcc_sptr self) -> pmt_vector_float

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

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

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

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

add_const_vcc_sptr.**set_k**(*add_const_vcc_sptr self*, *pmt_vector_cfloat k*)

Set additive constant vector.

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

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

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

gnuradio.blocks.**add_const_vff**(*pmt_vector_float k*) → add_const_vff_sptr

output[m] = input[m] + constant vector for all M streams.

Constructor Specific Documentation:

Create an instance of add_const_vff.

Parameters: **k** – additive constant vector

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

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

declare_sample_delay(add_const_vff_sptr self, unsigned int delay)

add_const_vff_sptr.**k**(*add_const_vff_sptr self*) → pmt_vector_float

Return additive constant vector.

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

swig_int_ptr

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

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

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

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

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

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

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

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

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

add_const_vff_sptr.**set_k**(*add_const_vff_sptr self*, *pmt_vector_float k*)
Set additive constant vector.

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

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

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

gnuradio.blocks.**add_const_vii**(*std::vector< int, std::allocator< int > > k*) → add_const_vii_sptr
output[m] = input[m] + constant vector for all M streams.

Constructor Specific Documentation:

Create an instance of add_const_vii.

Parameters: **k** – additive constant vector

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

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

add_const_vii_sptr.**k**(*add_const_vii_sptr self*) → std::vector< int,std::allocator< int > >
Return additive constant vector.

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

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

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

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

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

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

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

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

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

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

add_const_vii_sptr.**set_k**(*add_const_vii_sptr self*, *std::vector< int, std::allocator< int > > k*)
Set additive constant vector.

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

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

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

gnuradio.blocks.**add_const_vss**(*std::vector< short, std::allocator< short > > k*) → add_const_vss_sptr

output[m] = input[m] + constant vector for all M streams.

Constructor Specific Documentation:

Create an instance of add_const_vss.

Parameters: **k** – additive constant vector

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

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

declare_sample_delay(add_const_vss_sptr self, unsigned int delay)

add_const_vss_sptr.**k**(*add_const_vss_sptr self*) → std::vector< short,std::allocator< short > >

Return additive constant vector.

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

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

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

pc_input_buffers_full_avg(add_const_vss_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(add_const_vss_sptr self) -> pmt_vector_float

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

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

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

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

add_const_vss_sptr.**set_k**(*add_const_vss_sptr self*, *std::vector< short, std::allocator< short > > k*)

Set additive constant vector.

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

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

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

gnuradio.blocks.**add_ff**(*size_t vlen=1*) → add_ff_sptr

output = sum (input_0, input_1, ...)

Add across all input streams.

Constructor Specific Documentation:

Add streams of float values.

Parameters: **vlen** – Vector length

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

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

declare_sample_delay(add_ff_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(add_ff_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**add_ii**(*size_t vlen=1*) → add_ii_sptr
output = sum(input[0], input[1], ..., input[M-1])

Add samples across all input streams. For all samples on all input streams :

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**add_ss**(*size_t vlen=1*) → add_ss_sptr
output = sum(input[0], input[1], ..., input[M-1])

Add samples across all input streams. For all samples on all input streams :

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

pc_input_buffers_full_avg(add_ss_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**and_bb**(*size_t vlen=1*) → and_bb_sptr
output = input[0] & input[1] & ... & input[M-1]

bitwise boolean AND across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**and_const_bb**(*unsigned char k*) → and_const_bb_sptr
output[m] = input[m] & value for all M streams.

Bitwise boolean AND of constant with the data stream.

Constructor Specific Documentation:

Create an instance of and_const_bb.

Parameters: **k** – AND constant

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

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

and_const_bb_sptr.**k**(*and_const_bb_sptr self*) → unsigned char
Return AND constant.

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

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

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

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

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

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

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

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

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

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

and_const_bb_sptr.**set_k**(*and_const_bb_sptr self*, *unsigned char k*)
Set AND constant.

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

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

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

gnuradio.blocks.**and_const_ii**(*int k*) → and_const_ii_sptr
output[m] = input[m] & value for all M streams.

Bitwise boolean AND of constant with the data stream.

Constructor Specific Documentation:

Create an instance of and_const_ii.

Parameters: **k** – AND constant

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

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

and_const_ii_sptr.**k**(*and_const_ii_sptr self*) → int
Return AND constant.

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

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

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

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

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

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

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

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

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

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

and_const_ii_sptr.**set_k**(*and_const_ii_sptr self*, *int k*)

Set AND constant.

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

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

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

gnuradio.blocks.**and_const_ss**(*short k*) → and_const_ss_sptr

output[m] = input[m] & value for all M streams.

Bitwise boolean AND of constant with the data stream.

Constructor Specific Documentation:

Create an instance of and_const_ss.

Parameters: **k** – AND constant

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

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

declare_sample_delay(and_const_ss_sptr self, unsigned int delay)

and_const_ss_sptr.**k**(*and_const_ss_sptr self*) → short

Return AND constant.

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

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

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

pc_input_buffers_full_avg(and_const_ss_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(and_const_ss_sptr self) -> pmt_vector_float

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

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

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

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

and_const_ss_sptr.**set_k**(*and_const_ss_sptr self*, *short k*)

Set AND constant.

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

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

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

gnuradio.blocks.**and_ii**(*size_t vlen=1*) → and_ii_sptr

output = input[0] & input[1] & ... & input[M-1]

bitwise boolean AND across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

declare_sample_delay(and_ii_sptr self, unsigned int delay)

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**and_ss**(*size_t vlen=1*) → and_ss_sptr
output = input[0] & input[1] & ... & input[M-1]

bitwise boolean AND across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**annotator_1to1**(*int when*, *size_t sizeof_stream_item*) → annotator_1to1_sptr
1-to-1 stream annotator testing block. FOR TESTING PURPOSES ONLY.

This block creates tags to be sent downstream every 10,000 items it sees. The tags contain the name and ID of the instantiated block, use “seq” as a key, and have a counter that increments by 1 for every tag produced that is used as the tag’s value. The tags are propagated using the 1-to-1 policy.

It also stores a copy of all tags it sees flow past it. These tags can be recalled externally with the data() member.

Warning: This block is only meant for testing and showing how to use the tags.

Constructor Specific Documentation:

Parameters:
- **when** –
- **sizeof_stream_item** –

annotator_1to1_sptr.**active_thread_priority**(*annotator_1to1_sptr self*) → int

annotator_1to1_sptr.**data**(*annotator_1to1_sptr self*) → tags_vector_t

annotator_1to1_sptr.**declare_sample_delay**(*annotator_1to1_sptr self*, *int which*, *int delay*)
declare_sample_delay(annotator_1to1_sptr self, unsigned int delay)

annotator_1to1_sptr.**message_subscribers**(*annotator_1to1_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

annotator_1to1_sptr.**min_noutput_items**(*annotator_1to1_sptr self*) → int

annotator_1to1_sptr.**pc_input_buffers_full_avg**(*annotator_1to1_sptr self*, *int which*) → float
pc_input_buffers_full_avg(annotator_1to1_sptr self) -> pmt_vector_float

annotator_1to1_sptr.**pc_noutput_items_avg**(*annotator_1to1_sptr self*) → float

annotator_1to1_sptr.**pc_nproduced_avg**(*annotator_1to1_sptr self*) → float

annotator_1to1_sptr.**pc_output_buffers_full_avg**(*annotator_1to1_sptr self*, *int which*) → float
pc_output_buffers_full_avg(annotator_1to1_sptr self) -> pmt_vector_float

annotator_1to1_sptr.**pc_throughput_avg**(*annotator_1to1_sptr self*) → float

annotator_1to1_sptr.**pc_work_time_avg**(*annotator_1to1_sptr self*) → float

annotator_1to1_sptr.**pc_work_time_total**(*annotator_1to1_sptr self*) → float

annotator_1to1_sptr.**sample_delay**(*annotator_1to1_sptr self*, *int which*) → unsigned int

annotator_1to1_sptr.**set_min_noutput_items**(*annotator_1to1_sptr self*, *int m*)

annotator_1to1_sptr.**set_thread_priority**(*annotator_1to1_sptr self*, *int priority*) → int

annotator_1to1_sptr.**thread_priority**(*annotator_1to1_sptr self*) → int

gnuradio.blocks.**annotator_alltoall**(*int when*, *size_t sizeof_stream_item*) → annotator_alltoall_sptr
All-to-all stream annotator testing block. FOR TESTING PURPOSES ONLY.

This block creates tags to be sent downstream every 10,000 items it sees. The tags contain the name and ID of the instantiated block, use “seq” as a key, and have a counter that increments by 1 for every tag produced that is used as the tag’s value. The tags are propagated using the all-to-all policy.

It also stores a copy of all tags it sees flow past it. These tags can be recalled externally with the data() member.

This block is only meant for testing and showing how to use the tags.

Constructor Specific Documentation:

Parameters:
- **when** –
- **sizeof_stream_item** –

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

annotator_alltoall_sptr.**data**(*annotator_alltoall_sptr self*) → tags_vector_t

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

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

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

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

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

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

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

pc_output_buffers_full_avg(annotator_alltoall_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**annotator_raw**(*size_t sizeof_stream_item*) → annotator_raw_sptr

raw stream annotator testing block.

This block creates arbitrary tags to be sent downstream gnuradio/blocks to be sent are set manually via accessor methods and are sent only once.

This block is intended for testing of tag related gnuradio/blocks

Constructor Specific Documentation:

Parameters: **sizeof_stream_item** –

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

annotator_raw_sptr.**add_tag**(*annotator_raw_sptr self*, *uint64_t offset*, *swig_int_ptr key*, *swig_int_ptr val*)

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

declare_sample_delay(annotator_raw_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(annotator_raw_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(annotator_raw_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**argmax_fs**(*size_t vlen*) → argmax_fs_sptr

Compares vectors from multiple streams and determines the index in the vector and stream number where the maximum value occurred.

Data is passed in as a vector of length from multiple input sources. It will look through these streams of data items and output two streams:

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

declare_sample_delay(argmax_fs_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(argmax_fs_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(argmax_fs_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**argmax_is**(*size_t vlen*) → argmax_is_sptr

Compares vectors from multiple streams and determines the index in the vector and stream number where the maximum value occurred.

Data is passed in as a vector of length from multiple input sources. It will look through these streams of data items and output two streams:

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

declare_sample_delay(argmax_is_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(argmax_is_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(argmax_is_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**argmax_ss**(*size_t vlen*) → argmax_ss_sptr

Compares vectors from multiple streams and determines the index in the vector and stream number where the maximum value occurred.

Data is passed in as a vector of length from multiple input sources. It will look through these streams of data items and output two streams:

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

declare_sample_delay(argmax_ss_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(argmax_ss_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(argmax_ss_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**bin_statistics_f**(*unsigned int vlen*, *msg_queue_sptr msgq*, *gr::feval_dd * tune*, *size_t tune_delay*, *size_t dwell_delay*) → bin_statistics_f_sptr

control scanning and record frequency domain statistics

Constructor Specific Documentation:

Build a bin statistics block. See qa_bin_statistics.py and gr-uhd/examples/python/usrp_spectrum_sense.py for examples of its use, specifically how to use the callback function.

Parameters:

- **vlen** – vector length
- **msgq** – message queue
- **tune** – a feval_dd callback function
- **tune_delay** – number of samples for the tune delay
- **dwell_delay** – number of samples for the dwell delay

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

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

declare_sample_delay(bin_statistics_f_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(bin_statistics_f_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(bin_statistics_f_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**burst_tagger**(*size_t itemsize*) → burst_tagger_sptr

Sets a burst on/off tag based on the value of the trigger input.

This block takes two inputs, a signal stream on the input stream 0 and a trigger stream of shorts on input stream 1. If the trigger stream goes above 0, a tag with the key “burst” will be transmitted as a pmt::PMT_T. When the trigger signal falls below 0, the “burst” tag will be transmitted as pmt::PMT_F.

The signal on stream 0 is retransmitted to output stream 0.

Constructor Specific Documentation:

Build a burst tagger gnuradio/blocks.

Parameters: **itemsize** – itemsize of the signal stream on input 0.

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

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

declare_sample_delay(burst_tagger_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(burst_tagger_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(burst_tagger_sptr self) -> pmt_vector_float

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

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

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

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

burst_tagger_sptr.**set_false_tag**(*burst_tagger_sptr self*, *std::string const & key*, *bool value*)

For the false burst tag, change the key name to and a new value of .

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

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

burst_tagger_sptr.**set_true_tag**(*burst_tagger_sptr self*, *std::string const & key*, *bool value*)

For the true burst tag, change the key name to and a new value of .

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

gnuradio.blocks.**char_to_float**(*size_t vlen=1*, *float scale=1.0*) → char_to_float_sptr

Convert stream of chars to a stream of float.

Converts length vectors of input char samples to floats and applies a scaling factor of

Constructor Specific Documentation:

Build a chars to float stream converter block.

Parameters:
- **vlen** – vector length of data streams.
- **scale** – a scalar divider to change the output signal scale.

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

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

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

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

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

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

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

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

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

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

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

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

char_to_float_sptr.**scale**(*char_to_float_sptr self*) → float
Get the scalar divider value.

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

char_to_float_sptr.**set_scale**(*char_to_float_sptr self*, *float scale*)
Set the scalar divider value.

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

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

gnuradio.blocks.**char_to_short**(*size_t vlen=1*) → char_to_short_sptr
Convert stream of chars to a stream of shorts.

Converts length vectors of input char samples to shorts, multiplying each element by 256:

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**check_lfsr_32k_s**() → check_lfsr_32k_s_sptr

sink that checks if its input stream consists of a lfsr_32k sequence.

This sink is typically used along with gr::blocks::lfsr_32k_source_s to test the USRP using its digital loopback mode.

Constructor Specific Documentation:

check_lfsr_32k_s_sptr.**active_thread_priority**(*check_lfsr_32k_s_sptr self*) → int

check_lfsr_32k_s_sptr.**declare_sample_delay**(*check_lfsr_32k_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(check_lfsr_32k_s_sptr self, unsigned int delay)

check_lfsr_32k_s_sptr.**message_subscribers**(*check_lfsr_32k_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

check_lfsr_32k_s_sptr.**min_noutput_items**(*check_lfsr_32k_s_sptr self*) → int

check_lfsr_32k_s_sptr.**nright**(*check_lfsr_32k_s_sptr self*) → long

check_lfsr_32k_s_sptr.**ntotal**(*check_lfsr_32k_s_sptr self*) → long

check_lfsr_32k_s_sptr.**pc_input_buffers_full_avg**(*check_lfsr_32k_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(check_lfsr_32k_s_sptr self) -> pmt_vector_float

check_lfsr_32k_s_sptr.**pc_noutput_items_avg**(*check_lfsr_32k_s_sptr self*) → float

check_lfsr_32k_s_sptr.**pc_nproduced_avg**(*check_lfsr_32k_s_sptr self*) → float

check_lfsr_32k_s_sptr.**pc_output_buffers_full_avg**(*check_lfsr_32k_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(check_lfsr_32k_s_sptr self) -> pmt_vector_float

check_lfsr_32k_s_sptr.**pc_throughput_avg**(*check_lfsr_32k_s_sptr self*) → float

check_lfsr_32k_s_sptr.**pc_work_time_avg**(*check_lfsr_32k_s_sptr self*) → float

check_lfsr_32k_s_sptr.**pc_work_time_total**(*check_lfsr_32k_s_sptr self*) → float

check_lfsr_32k_s_sptr.**runlength**(*check_lfsr_32k_s_sptr self*) → long

check_lfsr_32k_s_sptr.**sample_delay**(*check_lfsr_32k_s_sptr self*, *int which*) → unsigned int

check_lfsr_32k_s_sptr.**set_min_noutput_items**(*check_lfsr_32k_s_sptr self*, *int m*)

check_lfsr_32k_s_sptr.**set_thread_priority**(*check_lfsr_32k_s_sptr self*, *int priority*) → int

check_lfsr_32k_s_sptr.**thread_priority**(*check_lfsr_32k_s_sptr self*) → int

gnuradio.blocks.**complex_to_arg**(*size_t vlen=1*) → complex_to_arg_sptr

complex in, arg (arctan) out (float)

Constructor Specific Documentation:

Build a complex to arg block.

Parameters: **vlen** – vector len (default 1)

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

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

declare_sample_delay(complex_to_arg_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(complex_to_arg_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**complex_to_float**(*size_t vlen=1*) → complex_to_float_sptr

Convert a stream of gr_complex to 1 or 2 streams of float.

If a single output stream is attached, this will output the real part of the input complex samples. If a second output stream is connected, output[0] is the real part and output[1] is the imaginary part.

Constructor Specific Documentation:

Build a complex to float block.

Parameters: **vlen** – vector len (default 1)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**complex_to_imag**(*size_t vlen=1*) → complex_to_imag_sptr

Produces the imaginary part (as a float0 of a complex stream.

Constructor Specific Documentation:

Build a complex to imaginary part block.

Parameters: **vlen** – vector len (default 1)

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

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

declare_sample_delay(complex_to_imag_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(complex_to_imag_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(complex_to_imag_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**complex_to_interleaved_short**(*bool vector=False*) → complex_to_interleaved_short_sptr

Convert stream of complex to a stream of interleaved shorts.

The output stream contains shorts with twice as many output items as input items. For every complex input item, we produce two output shorts that contain the real part and imaginary part converted to shorts:

Constructor Specific Documentation:

Build a complex to interleaved shorts block.

Parameters: **vector** –

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

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

declare_sample_delay(complex_to_interleaved_short_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(complex_to_interleaved_short_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(complex_to_interleaved_short_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**complex_to_mag**(*size_t vlen=1*) → complex_to_mag_sptr

complex in, magnitude out (float)

Calculates the magnitude of the complex samples:

Or: The input stream can be a vector of length , and for each vector, each item is converted using the above function. So above, m is from 0 to noutput_items*vlen for each call to work.

Constructor Specific Documentation:

Build a complex to magnitude block.

Parameters: **vlen** – vector len (default 1)

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

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

declare_sample_delay(complex_to_mag_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(complex_to_mag_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(complex_to_mag_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**complex_to_mag_squared**(*size_t vlen=1*) → complex_to_mag_squared_sptr

complex in, magnitude squared out (float)

Calculates the magnitude squared of the complex samples:

Or: The input stream can be a vector of length , and for each vector, each item is converted using the above function. So above, m is from 0 to noutput_items*vlen for each call to work.

Constructor Specific Documentation:

Build a complex to magnitude squared block.

Parameters: **vlen** – vector len (default 1)

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

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

declare_sample_delay(complex_to_mag_squared_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(complex_to_mag_squared_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(complex_to_mag_squared_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**complex_to_real**(*size_t vlen=1*) → complex_to_real_sptr

Produces the real part (as a float0 of a complex stream.

Constructor Specific Documentation:

Build a complex to real part block.

Parameters: **vlen** – vector len (default 1)

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

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

declare_sample_delay(complex_to_real_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(complex_to_real_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(complex_to_real_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**conjugate_cc**() → conjugate_cc_sptr

output = complex conjugate of input

Constructor Specific Documentation:

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

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

declare_sample_delay(conjugate_cc_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(conjugate_cc_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(conjugate_cc_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**copy**(*size_t itemsize*) → copy_sptr

output[i] = input[i]

When enabled (default), this block copies its input to its output. When disabled, this block drops its input on the floor.

Message Ports:

Constructor Specific Documentation:

Parameters: **itemsize** –

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

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

declare_sample_delay(copy_sptr self, unsigned int delay)

copy_sptr.**enabled**(*copy_sptr self*) → bool

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

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

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

pc_input_buffers_full_avg(copy_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(copy_sptr self) -> pmt_vector_float

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

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

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

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

copy_sptr.**set_enabled**(*copy_sptr self*, *bool enable*)

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

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

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

gnuradio.blocks.**ctrlport_probe2_b**(*std::string const & id*, *std::string const & desc*, *int len*, *unsigned int disp_mask*) → ctrlport_probe2_b_sptr

A ControlPort probe to export vectors of signals.

This block acts as a sink in the flowgraph but also exports vectors of complex samples over ControlPort. This block holds the latest number of complex samples so that every query by a ControlPort client will get the same length vector.

Constructor Specific Documentation:

Make a ControlPort probe block.

Parameters:
- **id** – A string ID to name the probe over ControlPort.
- **desc** – A string describing the probe.
- **len** – Number of samples to transmit.
- **disp_mask** – Mask to set default display params.

ctrlport_probe2_b_sptr.**active_thread_priority**(*ctrlport_probe2_b_sptr self*) → int

ctrlport_probe2_b_sptr.**declare_sample_delay**(*ctrlport_probe2_b_sptr self*, *int which*, *int delay*)

declare_sample_delay(ctrlport_probe2_b_sptr self, unsigned int delay)

ctrlport_probe2_b_sptr.**get**(*ctrlport_probe2_b_sptr self*) → std::vector< signed char,std::allocator< signed char > >

ctrlport_probe2_b_sptr.**message_subscribers**(*ctrlport_probe2_b_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

ctrlport_probe2_b_sptr.**min_noutput_items**(*ctrlport_probe2_b_sptr self*) → int

ctrlport_probe2_b_sptr.**pc_input_buffers_full_avg**(*ctrlport_probe2_b_sptr self*, *int which*) → float

pc_input_buffers_full_avg(ctrlport_probe2_b_sptr self) -> pmt_vector_float

ctrlport_probe2_b_sptr.**pc_noutput_items_avg**(*ctrlport_probe2_b_sptr self*) → float

ctrlport_probe2_b_sptr.**pc_nproduced_avg**(*ctrlport_probe2_b_sptr self*) → float

ctrlport_probe2_b_sptr.**pc_output_buffers_full_avg**(*ctrlport_probe2_b_sptr self*, *int which*) → float

pc_output_buffers_full_avg(ctrlport_probe2_b_sptr self) -> pmt_vector_float

ctrlport_probe2_b_sptr.**pc_throughput_avg**(*ctrlport_probe2_b_sptr self*) → float

ctrlport_probe2_b_sptr.**pc_work_time_avg**(*ctrlport_probe2_b_sptr self*) → float

ctrlport_probe2_b_sptr.**pc_work_time_total**(*ctrlport_probe2_b_sptr self*) → float

ctrlport_probe2_b_sptr.**sample_delay**(*ctrlport_probe2_b_sptr self*, *int which*) → unsigned int

ctrlport_probe2_b_sptr.**set_length**(*ctrlport_probe2_b_sptr self*, *int len*)

ctrlport_probe2_b_sptr.**set_min_noutput_items**(*ctrlport_probe2_b_sptr self*, *int m*)

ctrlport_probe2_b_sptr.**set_thread_priority**(*ctrlport_probe2_b_sptr self*, *int priority*) → int

ctrlport_probe2_b_sptr.**thread_priority**(*ctrlport_probe2_b_sptr self*) → int

gnuradio.blocks.**ctrlport_probe2_c**(*std::string const & id*, *std::string const & desc*, *int len*, *unsigned int disp_mask*) → ctrlport_probe2_c_sptr

A ControlPort probe to export vectors of signals.

This block acts as a sink in the flowgraph but also exports vectors of complex samples over ControlPort. This block holds the latest number of complex samples so that every query by a ControlPort client will get the same

length vector.

Constructor Specific Documentation:

Make a ControlPort probe block.

Parameters:
- **id** – A string ID to name the probe over ControlPort.
- **desc** – A string describing the probe.
- **len** – Number of samples to transmit.
- **disp_mask** – Mask to set default display params.

ctrlport_probe2_c_sptr.**active_thread_priority**(*ctrlport_probe2_c_sptr self*) → int

ctrlport_probe2_c_sptr.**declare_sample_delay**(*ctrlport_probe2_c_sptr self*, *int which*, *int delay*)
declare_sample_delay(ctrlport_probe2_c_sptr self, unsigned int delay)

ctrlport_probe2_c_sptr.**get**(*ctrlport_probe2_c_sptr self*) → pmt_vector_cfloat

ctrlport_probe2_c_sptr.**message_subscribers**(*ctrlport_probe2_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

ctrlport_probe2_c_sptr.**min_noutput_items**(*ctrlport_probe2_c_sptr self*) → int

ctrlport_probe2_c_sptr.**pc_input_buffers_full_avg**(*ctrlport_probe2_c_sptr self*, *int which*) → float
pc_input_buffers_full_avg(ctrlport_probe2_c_sptr self) -> pmt_vector_float

ctrlport_probe2_c_sptr.**pc_noutput_items_avg**(*ctrlport_probe2_c_sptr self*) → float

ctrlport_probe2_c_sptr.**pc_nproduced_avg**(*ctrlport_probe2_c_sptr self*) → float

ctrlport_probe2_c_sptr.**pc_output_buffers_full_avg**(*ctrlport_probe2_c_sptr self*, *int which*) → float
pc_output_buffers_full_avg(ctrlport_probe2_c_sptr self) -> pmt_vector_float

ctrlport_probe2_c_sptr.**pc_throughput_avg**(*ctrlport_probe2_c_sptr self*) → float

ctrlport_probe2_c_sptr.**pc_work_time_avg**(*ctrlport_probe2_c_sptr self*) → float

ctrlport_probe2_c_sptr.**pc_work_time_total**(*ctrlport_probe2_c_sptr self*) → float

ctrlport_probe2_c_sptr.**sample_delay**(*ctrlport_probe2_c_sptr self*, *int which*) → unsigned int

ctrlport_probe2_c_sptr.**set_length**(*ctrlport_probe2_c_sptr self*, *int len*)

ctrlport_probe2_c_sptr.**set_min_noutput_items**(*ctrlport_probe2_c_sptr self*, *int m*)

ctrlport_probe2_c_sptr.**set_thread_priority**(*ctrlport_probe2_c_sptr self*, *int priority*) → int

ctrlport_probe2_c_sptr.**thread_priority**(*ctrlport_probe2_c_sptr self*) → int

gnuradio.blocks.**ctrlport_probe2_f**(*std::string const & id*, *std::string const & desc*, *int len*, *unsigned int disp_mask*) → ctrlport_probe2_f_sptr
A ControlPort probe to export vectors of signals.

This block acts as a sink in the flowgraph but also exports vectors of complex samples over ControlPort. This block holds the latest number of complex samples so that every query by a ControlPort client will get the same length vector.

Constructor Specific Documentation:

Make a ControlPort probe block.

Parameters:
- **id** – A string ID to name the probe over ControlPort.
- **desc** – A string describing the probe.
- **len** – Number of samples to transmit.
- **disp_mask** – Mask to set default display params.

ctrlport_probe2_f_sptr.**active_thread_priority**(*ctrlport_probe2_f_sptr self*) → int

ctrlport_probe2_f_sptr.**declare_sample_delay**(*ctrlport_probe2_f_sptr self*, *int which*, *int delay*)
declare_sample_delay(ctrlport_probe2_f_sptr self, unsigned int delay)

ctrlport_probe2_f_sptr.**get**(*ctrlport_probe2_f_sptr self*) → pmt_vector_float

ctrlport_probe2_f_sptr.**message_subscribers**(*ctrlport_probe2_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

ctrlport_probe2_f_sptr.**min_noutput_items**(*ctrlport_probe2_f_sptr self*) → int

ctrlport_probe2_f_sptr.**pc_input_buffers_full_avg**(*ctrlport_probe2_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(ctrlport_probe2_f_sptr self) -> pmt_vector_float

ctrlport_probe2_f_sptr.**pc_noutput_items_avg**(*ctrlport_probe2_f_sptr self*) → float

ctrlport_probe2_f_sptr.**pc_nproduced_avg**(*ctrlport_probe2_f_sptr self*) → float

ctrlport_probe2_f_sptr.**pc_output_buffers_full_avg**(*ctrlport_probe2_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(ctrlport_probe2_f_sptr self) -> pmt_vector_float

ctrlport_probe2_f_sptr.**pc_throughput_avg**(*ctrlport_probe2_f_sptr self*) → float

ctrlport_probe2_f_sptr.**pc_work_time_avg**(*ctrlport_probe2_f_sptr self*) → float

ctrlport_probe2_f_sptr.**pc_work_time_total**(*ctrlport_probe2_f_sptr self*) → float

ctrlport_probe2_f_sptr.**sample_delay**(*ctrlport_probe2_f_sptr self*, *int which*) → unsigned int

ctrlport_probe2_f_sptr.**set_length**(*ctrlport_probe2_f_sptr self*, *int len*)

ctrlport_probe2_f_sptr.**set_min_noutput_items**(*ctrlport_probe2_f_sptr self*, *int m*)

ctrlport_probe2_f_sptr.**set_thread_priority**(*ctrlport_probe2_f_sptr self*, *int priority*) → int

ctrlport_probe2_f_sptr.**thread_priority**(*ctrlport_probe2_f_sptr self*) → int

gnuradio.blocks.**ctrlport_probe2_i**(*std::string const & id*, *std::string const & desc*, *int len*, *unsigned int disp_mask*) → ctrlport_probe2_i_sptr

A ControlPort probe to export vectors of signals.

This block acts as a sink in the flowgraph but also exports vectors of complex samples over ControlPort. This block holds the latest number of complex samples so that every query by a ControlPort client will get the same length vector.

Constructor Specific Documentation:

Make a ControlPort probe block.

Parameters:
- **id** – A string ID to name the probe over ControlPort.
- **desc** – A string describing the probe.
- **len** – Number of samples to transmit.
- **disp_mask** – Mask to set default display params.

ctrlport_probe2_i_sptr.**active_thread_priority**(*ctrlport_probe2_i_sptr self*) → int

ctrlport_probe2_i_sptr.**declare_sample_delay**(*ctrlport_probe2_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(ctrlport_probe2_i_sptr self, unsigned int delay)

ctrlport_probe2_i_sptr.**get**(*ctrlport_probe2_i_sptr self*) → std::vector< int,std::allocator< int > >

ctrlport_probe2_i_sptr.**message_subscribers**(*ctrlport_probe2_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

ctrlport_probe2_i_sptr.**min_noutput_items**(*ctrlport_probe2_i_sptr self*) → int

ctrlport_probe2_i_sptr.**pc_input_buffers_full_avg**(*ctrlport_probe2_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(ctrlport_probe2_i_sptr self) -> pmt_vector_float

ctrlport_probe2_i_sptr.**pc_noutput_items_avg**(*ctrlport_probe2_i_sptr self*) → float

ctrlport_probe2_i_sptr.**pc_nproduced_avg**(*ctrlport_probe2_i_sptr self*) → float

ctrlport_probe2_i_sptr.**pc_output_buffers_full_avg**(*ctrlport_probe2_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(ctrlport_probe2_i_sptr self) -> pmt_vector_float

ctrlport_probe2_i_sptr.**pc_throughput_avg**(*ctrlport_probe2_i_sptr self*) → float

ctrlport_probe2_i_sptr.**pc_work_time_avg**(*ctrlport_probe2_i_sptr self*) → float

ctrlport_probe2_i_sptr.**pc_work_time_total**(*ctrlport_probe2_i_sptr self*) → float

ctrlport_probe2_i_sptr.**sample_delay**(*ctrlport_probe2_i_sptr self*, *int which*) → unsigned int

ctrlport_probe2_i_sptr.**set_length**(*ctrlport_probe2_i_sptr self*, *int len*)

ctrlport_probe2_i_sptr.**set_min_noutput_items**(*ctrlport_probe2_i_sptr self*, *int m*)

ctrlport_probe2_i_sptr.**set_thread_priority**(*ctrlport_probe2_i_sptr self*, *int priority*) → int

ctrlport_probe2_i_sptr.**thread_priority**(*ctrlport_probe2_i_sptr self*) → int

gnuradio.blocks.**ctrlport_probe2_s**(*std::string const & id*, *std::string const & desc*, *int len*, *unsigned int disp_mask*) → ctrlport_probe2_s_sptr

A ControlPort probe to export vectors of signals.

This block acts as a sink in the flowgraph but also exports vectors of complex samples over ControlPort. This block holds the latest number of complex samples so that every query by a ControlPort client will get the same length vector.

Constructor Specific Documentation:

Make a ControlPort probe block.

Parameters:
- **id** – A string ID to name the probe over ControlPort.
- **desc** – A string describing the probe.
- **len** – Number of samples to transmit.
- **disp_mask** – Mask to set default display params.

ctrlport_probe2_s_sptr.**active_thread_priority**(*ctrlport_probe2_s_sptr self*) → int

ctrlport_probe2_s_sptr.**declare_sample_delay**(*ctrlport_probe2_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(ctrlport_probe2_s_sptr self, unsigned int delay)

ctrlport_probe2_s_sptr.**get**(*ctrlport_probe2_s_sptr self*) → std::vector< short,std::allocator< short > >

ctrlport_probe2_s_sptr.**message_subscribers**(*ctrlport_probe2_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

ctrlport_probe2_s_sptr.**min_noutput_items**(*ctrlport_probe2_s_sptr self*) → int

ctrlport_probe2_s_sptr.**pc_input_buffers_full_avg**(*ctrlport_probe2_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(ctrlport_probe2_s_sptr self) -> pmt_vector_float

ctrlport_probe2_s_sptr.**pc_noutput_items_avg**(*ctrlport_probe2_s_sptr self*) → float

ctrlport_probe2_s_sptr.**pc_nproduced_avg**(*ctrlport_probe2_s_sptr self*) → float

ctrlport_probe2_s_sptr.**pc_output_buffers_full_avg**(*ctrlport_probe2_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(ctrlport_probe2_s_sptr self) -> pmt_vector_float

ctrlport_probe2_s_sptr.**pc_throughput_avg**(*ctrlport_probe2_s_sptr self*) → float

ctrlport_probe2_s_sptr.**pc_work_time_avg**(*ctrlport_probe2_s_sptr self*) → float

ctrlport_probe2_s_sptr.**pc_work_time_total**(*ctrlport_probe2_s_sptr self*) → float

ctrlport_probe2_s_sptr.**sample_delay**(*ctrlport_probe2_s_sptr self*, *int which*) → unsigned int

ctrlport_probe2_s_sptr.**set_length**(*ctrlport_probe2_s_sptr self*, *int len*)

ctrlport_probe2_s_sptr.**set_min_noutput_items**(*ctrlport_probe2_s_sptr self*, *int m*)

ctrlport_probe2_s_sptr.**set_thread_priority**(*ctrlport_probe2_s_sptr self*, *int priority*) → int

ctrlport_probe2_s_sptr.**thread_priority**(*ctrlport_probe2_s_sptr self*) → int

gnuradio.blocks.**ctrlport_probe_c**(*std::string const & id*, *std::string const & desc*) → ctrlport_probe_c_sptr

A ControlPort probe to export vectors of signals.

This block acts as a sink in the flowgraph but also exports vectors of complex samples over ControlPort. This block simply sends the current vector held in the work function when the queried by a ControlPort client.

Constructor Specific Documentation:

Make a ControlPort probe block.

Parameters:
- **id** – A string ID to name the probe over ControlPort.
- **desc** – A string describing the probe.

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

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

ctrlport_probe_c_sptr.**get**(*ctrlport_probe_c_sptr self*) → pmt_vector_cfloat

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**deinterleave**(*size_t itemsize*, *unsigned int blocksize=1*) → deinterleave_sptr
deinterleave an input block of samples into N outputs.

This block deinterleaves blocks of samples. For each output connection, the input stream will be deinterleaved successively to the output connections. By default, the block deinterleaves a single input to each output unless blocksize is given in the constructor.

Constructor Specific Documentation:

Make a deinterleave block.

Parameters:
- **itemsize** – stream itemsize
- **blocksize** – size of block to deinterleave

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**delay**(*size_t itemsize*, *int delay*) → delay_sptr

delay the input by a certain number of samples

Positive delays insert zero items at the beginning of the stream. Negative delays discard items from the stream.

You cannot initialize this block with a negative delay, however. That leads to a causality issue with the buffers when they are initialized. If you need to negetively delay one path, then put the positive delay on the other path instead.

Constructor Specific Documentation:

Make a delay block.

Parameters:
- **itemsize** – size of the data items.
- **delay** – number of samples to delay stream (>= 0).

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

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

declare_sample_delay(delay_sptr self, unsigned int delay)

delay_sptr.**dly**(*delay_sptr self*) → int

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

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

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

pc_input_buffers_full_avg(delay_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(delay_sptr self) -> pmt_vector_float

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

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

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

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

delay_sptr.**set_dly**(*delay_sptr self*, *int d*)

Reset the delay.

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

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

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

gnuradio.blocks.**divide_cc**(*size_t vlen=1*) → divide_cc_sptr

output = input[0] / input[1] / ... / input[M-1]

Divide across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

declare_sample_delay(divide_cc_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(divide_cc_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**divide_ff**(*size_t vlen=1*) → divide_ff_sptr
output = input[0] / input[1] / ... / input[M-1]

Divide across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**divide_ii**(*size_t vlen=1*) → divide_ii_sptr
output = input[0] / input[1] / ... / input[M-1]

Divide across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

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

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

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

pc_input_buffers_full_avg(divide_ii_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(divide_ii_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**divide_ss**(*size_t vlen=1*) → divide_ss_sptr

output = input[0] / input[1] / ... / input[M-1]

Divide across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

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

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

declare_sample_delay(divide_ss_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(divide_ss_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(divide_ss_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**endian_swap**(*size_t item_size_bytes=1*) → endian_swap_sptr

Convert stream of items into their byte swapped version.

Constructor Specific Documentation:

Make an endian swap block.

Parameters: **item_size_bytes** – number of bytes per item, 1=no-op, 2=uint16_t, 4=uint32_t, 8=uint64_t

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

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

declare_sample_delay(endian_swap_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(endian_swap_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(endian_swap_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**file_descriptor_sink**(*size_t itemsize*, *int fd*) → file_descriptor_sink_sptr

Write stream to file descriptor.

Constructor Specific Documentation:

Build a file descriptor sink block. The provided file descriptor will be closed when the sink is destroyed.

Parameters:
- **itemsize** – item size of the incoming data stream.
- **fd** – file descriptor (as an integer).

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

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

declare_sample_delay(file_descriptor_sink_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(file_descriptor_sink_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(file_descriptor_sink_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**file_descriptor_source**(*size_t itemsize*, *int fd*, *bool repeat=False*) → file_descriptor_source_sptr

Read stream from file descriptor.

Constructor Specific Documentation:

Build a file descriptor source block. The provided file descriptor will be closed when the sink is destroyed.

Parameters:
- **itemsize** – item size of the incoming data stream.
- **fd** – file descriptor (as an integer).
- **repeat** – repeat the data stream continuously.

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

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

declare_sample_delay(file_descriptor_source_sptr self, unsigned int delay)

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

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

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

pc_input_buffers_full_avg(file_descriptor_source_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(file_descriptor_source_sptr self) -> pmt_vector_float

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

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

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

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

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

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

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

gnuradio.blocks.**file_meta_sink**(*size_t itemsize*, *std::string const & filename*, *double samp_rate=1*, *double relative_rate=1*, *gr::blocks::gr_file_types type*, *bool complex=True*, *size_t max_segment_size=1000000*, *std::string const & extra_dict*, *bool detached_header=False*) → file_meta_sink_sptr

Write stream to file with meta-data headers.

These files represent data as binary information in between meta-data headers. The headers contain information about the type of data and properties of the data in the next segment of samples. The information includes:

Tags can be sent to the file to update the information, which will create a new header. Headers are found by searching from the first header (at position 0 in the file) and reading where the data segment starts plus the data segment size. Following will either be a new header or EOF.

Constructor Specific Documentation:

Create a meta-data file sink.

Parameters:
- **itemsize** – (size_t): Size of data type.
- **filename** – (string): Name of file to write data to.
- **samp_rate** – (double): Sample rate of data. If sample rate will be set by a tag, such as rx_tag from a UHD source, this is basically ignored.
- **relative_rate** – (double): Rate chance from source of sample rate tag to sink.
- **type** – (gr_file_types): Data type (int, float, etc.)
- **complex** – (bool): If data stream is complex
- **max_segment_size** – (size_t): Length of a single segment before the header is repeated (in items).
- **extra_dict** – (string): a serialized PMT dictionary of extra information. Currently not supported.
- **detached_header** – (bool): Set to true to store the header info in a separate file (named filename.hdr)

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

file_meta_sink_sptr.**close**(*file_meta_sink_sptr self*)

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

declare_sample_delay(file_meta_sink_sptr self, unsigned int delay)

file_meta_sink_sptr.**do_update**(*file_meta_sink_sptr self*)

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

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

file_meta_sink_sptr.**open**(*file_meta_sink_sptr self*, *std::string const & filename*) → bool

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

pc_input_buffers_full_avg(file_meta_sink_sptr self) -> pmt_vector_float

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

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

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

pc_output_buffers_full_avg(file_meta_sink_sptr self) -> pmt_vector_float

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

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

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

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

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

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

file_meta_sink_sptr.**set_unbuffered**(*file_meta_sink_sptr self*, *bool unbuffered*)

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

gnuradio.blocks.**file_meta_source**(*std::string const & filename*, *bool repeat=False*, *bool detached_header=False*, *std::string const & hdr_filename*) → file_meta_source_sptr

Reads stream from file with meta-data headers. Headers are parsed into tags.

The information in the metadata headers includes:

Any item inside of the extra header dictionary is ready out and made into a stream tag.

Constructor Specific Documentation:

Create a meta-data file source.

Parameters:
- **filename** – (string): Name of file to write data to.
- **repeat** – (bool): Repeats file when EOF is found.
- **detached_header** – (bool): Set to true if header info is stored in a separate file (usually named filename.hdr)
- **hdr_filename** – (string): Name of detached header file if used. Defaults to 'filename.hdr' if detached_header is true but this field is an empty string.

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

file_meta_source_sptr.**close**(*file_meta_source_sptr self*)

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

file_meta_source_sptr.**do_update**(*file_meta_source_sptr self*)

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

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

file_meta_source_sptr.**open**(*file_meta_source_sptr self*, *std::string const & filename*, *std::string const & hdr_filename*) → bool

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

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

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

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

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

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

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

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

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

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

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

gnuradio.blocks.**file_sink**(*size_t itemsize*, *char const * filename*, *bool append=False*) → file_sink_sptr
Write stream to file.

Constructor Specific Documentation:

Make a file sink.

Parameters:
- **itemsize** – size of the input data items.
- **filename** – name of the file to open and write output to.
- **append** – if true, data is appended to the file instead of overwriting the initial content.

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

file_sink_sptr.**close**(*file_sink_sptr self*)
Close current output file.

Closes current output file and ignores any output until open is called to connect to another file.

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

file_sink_sptr.**do_update**(*file_sink_sptr self*)
if we've had an update, do it now.

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

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

file_sink_sptr.**open**(*file_sink_sptr self*, *char const * filename*) → bool
Open filename and begin output to it.

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

file_sink_sptr.**pc_noutput_items_avg**(*file_sink_sptr self*) → float

file_sink_sptr.**pc_nproduced_avg**(*file_sink_sptr self*) → float

file_sink_sptr.**pc_output_buffers_full_avg**(*file_sink_sptr self*, *int which*) → float

pc_output_buffers_full_avg(file_sink_sptr self) -> pmt_vector_float

file_sink_sptr.**pc_throughput_avg**(*file_sink_sptr self*) → float

file_sink_sptr.**pc_work_time_avg**(*file_sink_sptr self*) → float

file_sink_sptr.**pc_work_time_total**(*file_sink_sptr self*) → float

file_sink_sptr.**sample_delay**(*file_sink_sptr self*, *int which*) → unsigned int

file_sink_sptr.**set_min_noutput_items**(*file_sink_sptr self*, *int m*)

file_sink_sptr.**set_thread_priority**(*file_sink_sptr self*, *int priority*) → int

file_sink_sptr.**set_unbuffered**(*file_sink_sptr self*, *bool unbuffered*)

turn on unbuffered writes for slower outputs

file_sink_sptr.**thread_priority**(*file_sink_sptr self*) → int

gnuradio.blocks.**file_source**(*size_t itemsize*, *char const * filename*, *bool repeat=False*) → file_source_sptr

Read stream from file.

Constructor Specific Documentation:

Create a file source.

Opens as a source of items into a flowgraph. The data is expected to be in binary format, item after item. The of the block determines the conversion from bits to items.

If is turned on, the file will repeat the file after it's reached the end.

Parameters:
- **itemsize** – the size of each item in the file, in bytes
- **filename** – name of the file to source from
- **repeat** – repeat file from start

file_source_sptr.**active_thread_priority**(*file_source_sptr self*) → int

file_source_sptr.**close**(*file_source_sptr self*)

Close the file handle.

file_source_sptr.**declare_sample_delay**(*file_source_sptr self*, *int which*, *int delay*)

declare_sample_delay(file_source_sptr self, unsigned int delay)

file_source_sptr.**message_subscribers**(*file_source_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

file_source_sptr.**min_noutput_items**(*file_source_sptr self*) → int

file_source_sptr.**open**(*file_source_sptr self*, *char const * filename*, *bool repeat*)

Opens a new file.

file_source_sptr.**pc_input_buffers_full_avg**(*file_source_sptr self*, *int which*) → float

pc_input_buffers_full_avg(file_source_sptr self) -> pmt_vector_float

file_source_sptr.**pc_noutput_items_avg**(*file_source_sptr self*) → float

file_source_sptr.**pc_nproduced_avg**(*file_source_sptr self*) → float

file_source_sptr.**pc_output_buffers_full_avg**(*file_source_sptr self*, *int which*) → float

pc_output_buffers_full_avg(file_source_sptr self) -> pmt_vector_float

file_source_sptr.**pc_throughput_avg**(*file_source_sptr self*) → float

file_source_sptr.**pc_work_time_avg**(*file_source_sptr self*) → float

file_source_sptr.**pc_work_time_total**(*file_source_sptr self*) → float

file_source_sptr.**sample_delay**(*file_source_sptr self*, *int which*) → unsigned int

file_source_sptr.**seek**(*file_source_sptr self*, *long seek_point*, *int whence*) → bool

seek file to relative to

file_source_sptr.**set_begin_tag**(*file_source_sptr self*, *swig_int_ptr val*)

Add a stream tag to the first sample of the file if true.

file_source_sptr.**set_min_noutput_items**(*file_source_sptr self*, *int m*)

file_source_sptr.**set_thread_priority**(*file_source_sptr self*, *int priority*) → int

file_source_sptr.**thread_priority**(*file_source_sptr self*) → int

gnuradio.blocks.**float_to_char**(*size_t vlen=1*, *float scale=1.0*) → float_to_char_sptr

Convert stream of floats to a stream of char.

Constructor Specific Documentation:

Build a float to char block.

Parameters:
- **vlen** – vector length of data streams.
- **scale** – a scalar multiplier to change the output signal scale.

float_to_char_sptr.**active_thread_priority**(*float_to_char_sptr self*) → int

float_to_char_sptr.**declare_sample_delay**(*float_to_char_sptr self*, *int which*, *int delay*)

declare_sample_delay(float_to_char_sptr self, unsigned int delay)

float_to_char_sptr.**message_subscribers**(*float_to_char_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

float_to_char_sptr.**min_noutput_items**(*float_to_char_sptr self*) → int

float_to_char_sptr.**pc_input_buffers_full_avg**(*float_to_char_sptr self*, *int which*) → float

pc_input_buffers_full_avg(float_to_char_sptr self) -> pmt_vector_float

float_to_char_sptr.**pc_noutput_items_avg**(*float_to_char_sptr self*) → float

float_to_char_sptr.**pc_nproduced_avg**(*float_to_char_sptr self*) → float

float_to_char_sptr.**pc_output_buffers_full_avg**(*float_to_char_sptr self*, *int which*) → float

pc_output_buffers_full_avg(float_to_char_sptr self) -> pmt_vector_float

float_to_char_sptr.**pc_throughput_avg**(*float_to_char_sptr self*) → float

float_to_char_sptr.**pc_work_time_avg**(*float_to_char_sptr self*) → float

float_to_char_sptr.**pc_work_time_total**(*float_to_char_sptr self*) → float

float_to_char_sptr.**sample_delay**(*float_to_char_sptr self*, *int which*) → unsigned int

float_to_char_sptr.**scale**(*float_to_char_sptr self*) → float

Get the scalar multiplier value.

float_to_char_sptr.**set_min_noutput_items**(*float_to_char_sptr self*, *int m*)

float_to_char_sptr.**set_scale**(*float_to_char_sptr self*, *float scale*)

Set the scalar multiplier value.

float_to_char_sptr.**set_thread_priority**(*float_to_char_sptr self*, *int priority*) → int

float_to_char_sptr.**thread_priority**(*float_to_char_sptr self*) → int

gnuradio.blocks.**float_to_complex**(*size_t vlen=1*) → float_to_complex_sptr

one or two floats in, complex out

Constructor Specific Documentation:

Build a float to complex block.

Parameters: **vlen** – vector len (default 1)

float_to_complex_sptr.**active_thread_priority**(*float_to_complex_sptr self*) → int

float_to_complex_sptr.**declare_sample_delay**(*float_to_complex_sptr self*, *int which*, *int delay*)

declare_sample_delay(float_to_complex_sptr self, unsigned int delay)

float_to_complex_sptr.**message_subscribers**(*float_to_complex_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

float_to_complex_sptr.**min_noutput_items**(*float_to_complex_sptr self*) → int

float_to_complex_sptr.**pc_input_buffers_full_avg**(*float_to_complex_sptr self*, *int which*) → float

pc_input_buffers_full_avg(float_to_complex_sptr self) -> pmt_vector_float

float_to_complex_sptr.**pc_noutput_items_avg**(*float_to_complex_sptr self*) → float

float_to_complex_sptr.**pc_nproduced_avg**(*float_to_complex_sptr self*) → float

float_to_complex_sptr.**pc_output_buffers_full_avg**(*float_to_complex_sptr self*, *int which*) → float

pc_output_buffers_full_avg(float_to_complex_sptr self) -> pmt_vector_float

float_to_complex_sptr.**pc_throughput_avg**(*float_to_complex_sptr self*) → float

float_to_complex_sptr.**pc_work_time_avg**(*float_to_complex_sptr self*) → float

float_to_complex_sptr.**pc_work_time_total**(*float_to_complex_sptr self*) → float

float_to_complex_sptr.**sample_delay**(*float_to_complex_sptr self*, *int which*) → unsigned int

float_to_complex_sptr.**set_min_noutput_items**(*float_to_complex_sptr self*, *int m*)

float_to_complex_sptr.**set_thread_priority**(*float_to_complex_sptr self*, *int priority*) → int

float_to_complex_sptr.**thread_priority**(*float_to_complex_sptr self*) → int

gnuradio.blocks.**float_to_int**(*size_t vlen=1*, *float scale=1.0*) → float_to_int_sptr

Convert stream of floats to a stream of ints.

Constructor Specific Documentation:

Build a float to int block.

Parameters:
- **vlen** – vector length of data streams.
- **scale** – a scalar multiplier to change the output signal scale.

float_to_int_sptr.**active_thread_priority**(*float_to_int_sptr self*) → int

float_to_int_sptr.**declare_sample_delay**(*float_to_int_sptr self*, *int which*, *int delay*)

declare_sample_delay(float_to_int_sptr self, unsigned int delay)

float_to_int_sptr.**message_subscribers**(*float_to_int_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

float_to_int_sptr.**min_noutput_items**(*float_to_int_sptr self*) → int

float_to_int_sptr.**pc_input_buffers_full_avg**(*float_to_int_sptr self*, *int which*) → float

pc_input_buffers_full_avg(float_to_int_sptr self) -> pmt_vector_float

float_to_int_sptr.**pc_noutput_items_avg**(*float_to_int_sptr self*) → float

float_to_int_sptr.**pc_nproduced_avg**(*float_to_int_sptr self*) → float

float_to_int_sptr.**pc_output_buffers_full_avg**(*float_to_int_sptr self*, *int which*) → float

pc_output_buffers_full_avg(float_to_int_sptr self) -> pmt_vector_float

float_to_int_sptr.**pc_throughput_avg**(*float_to_int_sptr self*) → float

float_to_int_sptr.**pc_work_time_avg**(*float_to_int_sptr self*) → float

float_to_int_sptr.**pc_work_time_total**(*float_to_int_sptr self*) → float

float_to_int_sptr.**sample_delay**(*float_to_int_sptr self*, *int which*) → unsigned int

float_to_int_sptr.**scale**(*float_to_int_sptr self*) → float

Get the scalar multiplier value.

float_to_int_sptr.**set_min_noutput_items**(*float_to_int_sptr self*, *int m*)

float_to_int_sptr.**set_scale**(*float_to_int_sptr self*, *float scale*)

Set the scalar multiplier value.

float_to_int_sptr.**set_thread_priority**(*float_to_int_sptr self*, *int priority*) → int

float_to_int_sptr.**thread_priority**(*float_to_int_sptr self*) → int

gnuradio.blocks.**float_to_short**(*size_t vlen=1*, *float scale=1.0*) → float_to_short_sptr

Convert stream of floats to a stream of shorts.

Constructor Specific Documentation:

Build a float to short block.

Parameters:
- **vlen** – vector length of data streams.
- **scale** – a scalar multiplier to change the output signal scale.

float_to_short_sptr.**active_thread_priority**(*float_to_short_sptr self*) → int

float_to_short_sptr.**declare_sample_delay**(*float_to_short_sptr self*, *int which*, *int delay*)
declare_sample_delay(float_to_short_sptr self, unsigned int delay)

float_to_short_sptr.**message_subscribers**(*float_to_short_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

float_to_short_sptr.**min_noutput_items**(*float_to_short_sptr self*) → int

float_to_short_sptr.**pc_input_buffers_full_avg**(*float_to_short_sptr self*, *int which*) → float
pc_input_buffers_full_avg(float_to_short_sptr self) -> pmt_vector_float

float_to_short_sptr.**pc_noutput_items_avg**(*float_to_short_sptr self*) → float

float_to_short_sptr.**pc_nproduced_avg**(*float_to_short_sptr self*) → float

float_to_short_sptr.**pc_output_buffers_full_avg**(*float_to_short_sptr self*, *int which*) → float
pc_output_buffers_full_avg(float_to_short_sptr self) -> pmt_vector_float

float_to_short_sptr.**pc_throughput_avg**(*float_to_short_sptr self*) → float

float_to_short_sptr.**pc_work_time_avg**(*float_to_short_sptr self*) → float

float_to_short_sptr.**pc_work_time_total**(*float_to_short_sptr self*) → float

float_to_short_sptr.**sample_delay**(*float_to_short_sptr self*, *int which*) → unsigned int

float_to_short_sptr.**scale**(*float_to_short_sptr self*) → float
Get the scalar multiplier value.

float_to_short_sptr.**set_min_noutput_items**(*float_to_short_sptr self*, *int m*)

float_to_short_sptr.**set_scale**(*float_to_short_sptr self*, *float scale*)
Set the scalar multiplier value.

float_to_short_sptr.**set_thread_priority**(*float_to_short_sptr self*, *int priority*) → int

float_to_short_sptr.**thread_priority**(*float_to_short_sptr self*) → int

gnuradio.blocks.**float_to_uchar**() → float_to_uchar_sptr
Convert stream of floats to a stream of unsigned chars.

Constructor Specific Documentation:

Build a float to uchar block.

float_to_uchar_sptr.**active_thread_priority**(*float_to_uchar_sptr self*) → int

float_to_uchar_sptr.**declare_sample_delay**(*float_to_uchar_sptr self*, *int which*, *int delay*)
declare_sample_delay(float_to_uchar_sptr self, unsigned int delay)

float_to_uchar_sptr.**message_subscribers**(*float_to_uchar_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

float_to_uchar_sptr.**min_noutput_items**(*float_to_uchar_sptr self*) → int

float_to_uchar_sptr.**pc_input_buffers_full_avg**(*float_to_uchar_sptr self*, *int which*) → float
pc_input_buffers_full_avg(float_to_uchar_sptr self) -> pmt_vector_float

float_to_uchar_sptr.**pc_noutput_items_avg**(*float_to_uchar_sptr self*) → float

float_to_uchar_sptr.**pc_nproduced_avg**(*float_to_uchar_sptr self*) → float

float_to_uchar_sptr.**pc_output_buffers_full_avg**(*float_to_uchar_sptr self*, *int which*) → float
pc_output_buffers_full_avg(float_to_uchar_sptr self) -> pmt_vector_float

float_to_uchar_sptr.**pc_throughput_avg**(*float_to_uchar_sptr self*) → float

float_to_uchar_sptr.**pc_work_time_avg**(*float_to_uchar_sptr self*) → float

float_to_uchar_sptr.**pc_work_time_total**(*float_to_uchar_sptr self*) → float

float_to_uchar_sptr.**sample_delay**(*float_to_uchar_sptr self*, *int which*) → unsigned int

float_to_uchar_sptr.**set_min_noutput_items**(*float_to_uchar_sptr self*, *int m*)

float_to_uchar_sptr.**set_thread_priority**(*float_to_uchar_sptr self*, *int priority*) → int

float_to_uchar_sptr.**thread_priority**(*float_to_uchar_sptr self*) → int

gnuradio.blocks.**head**(*size_t sizeof_stream_item*, *uint64_t nitems*) → head_sptr

copies the first N items to the output then signals done

Useful for building test cases

Constructor Specific Documentation:

Parameters:
- **sizeof_stream_item** –
- **nitems** –

head_sptr.**active_thread_priority**(*head_sptr self*) → int

head_sptr.**declare_sample_delay**(*head_sptr self*, *int which*, *int delay*)

declare_sample_delay(head_sptr self, unsigned int delay)

head_sptr.**message_subscribers**(*head_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

head_sptr.**min_noutput_items**(*head_sptr self*) → int

head_sptr.**pc_input_buffers_full_avg**(*head_sptr self*, *int which*) → float

pc_input_buffers_full_avg(head_sptr self) -> pmt_vector_float

head_sptr.**pc_noutput_items_avg**(*head_sptr self*) → float

head_sptr.**pc_nproduced_avg**(*head_sptr self*) → float

head_sptr.**pc_output_buffers_full_avg**(*head_sptr self*, *int which*) → float

pc_output_buffers_full_avg(head_sptr self) -> pmt_vector_float

head_sptr.**pc_throughput_avg**(*head_sptr self*) → float

head_sptr.**pc_work_time_avg**(*head_sptr self*) → float

head_sptr.**pc_work_time_total**(*head_sptr self*) → float

head_sptr.**reset**(*head_sptr self*)

head_sptr.**sample_delay**(*head_sptr self*, *int which*) → unsigned int

head_sptr.**set_length**(*head_sptr self*, *uint64_t nitems*)

head_sptr.**set_min_noutput_items**(*head_sptr self*, *int m*)

head_sptr.**set_thread_priority**(*head_sptr self*, *int priority*) → int

head_sptr.**thread_priority**(*head_sptr self*) → int

gnuradio.blocks.**int_to_float**(*size_t vlen=1*, *float scale=1.0*) → int_to_float_sptr

Convert stream of ints to a stream of floats.

Constructor Specific Documentation:

Build an int to float block.

Parameters:
- **vlen** – vector length of data streams.
- **scale** – a scalar divider to change the output signal scale.

int_to_float_sptr.**active_thread_priority**(*int_to_float_sptr self*) → int

int_to_float_sptr.**declare_sample_delay**(*int_to_float_sptr self*, *int which*, *int delay*)

declare_sample_delay(int_to_float_sptr self, unsigned int delay)

int_to_float_sptr.**message_subscribers**(*int_to_float_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

int_to_float_sptr.**min_noutput_items**(*int_to_float_sptr self*) → int

int_to_float_sptr.**pc_input_buffers_full_avg**(*int_to_float_sptr self*, *int which*) → float

pc_input_buffers_full_avg(int_to_float_sptr self) -> pmt_vector_float

int_to_float_sptr.**pc_noutput_items_avg**(*int_to_float_sptr self*) → float

int_to_float_sptr.**pc_nproduced_avg**(*int_to_float_sptr self*) → float

int_to_float_sptr.**pc_output_buffers_full_avg**(*int_to_float_sptr self*, *int which*) → float

pc_output_buffers_full_avg(int_to_float_sptr self) -> pmt_vector_float

int_to_float_sptr.**pc_throughput_avg**(*int_to_float_sptr self*) → float

int_to_float_sptr.**pc_work_time_avg**(*int_to_float_sptr self*) → float

int_to_float_sptr.**pc_work_time_total**(*int_to_float_sptr self*) → float

int_to_float_sptr.**sample_delay**(*int_to_float_sptr self*, *int which*) → unsigned int

int_to_float_sptr.**scale**(*int_to_float_sptr self*) → float

Get the scalar divider value.

int_to_float_sptr.**set_min_noutput_items**(*int_to_float_sptr self*, *int m*)

int_to_float_sptr.**set_scale**(*int_to_float_sptr self*, *float scale*)

Set the scalar divider value.

int_to_float_sptr.**set_thread_priority**(*int_to_float_sptr self*, *int priority*) → int

int_to_float_sptr.**thread_priority**(*int_to_float_sptr self*) → int

gnuradio.blocks.**integrate_cc**(*int decim*, *int vlen=1*) → integrate_cc_sptr

Integrate successive samples and decimate.

Constructor Specific Documentation:

Parameters:
- **decim** –
- **vlen** –

integrate_cc_sptr.**active_thread_priority**(*integrate_cc_sptr self*) → int

integrate_cc_sptr.**declare_sample_delay**(*integrate_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(integrate_cc_sptr self, unsigned int delay)

integrate_cc_sptr.**message_subscribers**(*integrate_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

integrate_cc_sptr.**min_noutput_items**(*integrate_cc_sptr self*) → int

integrate_cc_sptr.**pc_input_buffers_full_avg**(*integrate_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(integrate_cc_sptr self) -> pmt_vector_float

integrate_cc_sptr.**pc_noutput_items_avg**(*integrate_cc_sptr self*) → float

integrate_cc_sptr.**pc_nproduced_avg**(*integrate_cc_sptr self*) → float

integrate_cc_sptr.**pc_output_buffers_full_avg**(*integrate_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(integrate_cc_sptr self) -> pmt_vector_float

integrate_cc_sptr.**pc_throughput_avg**(*integrate_cc_sptr self*) → float

integrate_cc_sptr.**pc_work_time_avg**(*integrate_cc_sptr self*) → float

integrate_cc_sptr.**pc_work_time_total**(*integrate_cc_sptr self*) → float

integrate_cc_sptr.**sample_delay**(*integrate_cc_sptr self*, *int which*) → unsigned int

integrate_cc_sptr.**set_min_noutput_items**(*integrate_cc_sptr self*, *int m*)

integrate_cc_sptr.**set_thread_priority**(*integrate_cc_sptr self*, *int priority*) → int

integrate_cc_sptr.**thread_priority**(*integrate_cc_sptr self*) → int

gnuradio.blocks.**integrate_ff**(*int decim*, *int vlen=1*) → integrate_ff_sptr

Integrate successive samples and decimate.

Constructor Specific Documentation:

Parameters:
- **decim** –
- **vlen** –

integrate_ff_sptr.**active_thread_priority**(*integrate_ff_sptr self*) → int

integrate_ff_sptr.**declare_sample_delay**(*integrate_ff_sptr self*, *int which*, *int delay*)
declare_sample_delay(integrate_ff_sptr self, unsigned int delay)

integrate_ff_sptr.**message_subscribers**(*integrate_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

integrate_ff_sptr.**min_noutput_items**(*integrate_ff_sptr self*) → int

integrate_ff_sptr.**pc_input_buffers_full_avg**(*integrate_ff_sptr self*, *int which*) → float
pc_input_buffers_full_avg(integrate_ff_sptr self) -> pmt_vector_float

integrate_ff_sptr.**pc_noutput_items_avg**(*integrate_ff_sptr self*) → float

integrate_ff_sptr.**pc_nproduced_avg**(*integrate_ff_sptr self*) → float

integrate_ff_sptr.**pc_output_buffers_full_avg**(*integrate_ff_sptr self*, *int which*) → float
pc_output_buffers_full_avg(integrate_ff_sptr self) -> pmt_vector_float

integrate_ff_sptr.**pc_throughput_avg**(*integrate_ff_sptr self*) → float

integrate_ff_sptr.**pc_work_time_avg**(*integrate_ff_sptr self*) → float

integrate_ff_sptr.**pc_work_time_total**(*integrate_ff_sptr self*) → float

integrate_ff_sptr.**sample_delay**(*integrate_ff_sptr self*, *int which*) → unsigned int

integrate_ff_sptr.**set_min_noutput_items**(*integrate_ff_sptr self*, *int m*)

integrate_ff_sptr.**set_thread_priority**(*integrate_ff_sptr self*, *int priority*) → int

integrate_ff_sptr.**thread_priority**(*integrate_ff_sptr self*) → int

gnuradio.blocks.**integrate_ii**(*int decim*, *int vlen=1*) → integrate_ii_sptr
Integrate successive samples and decimate.

Constructor Specific Documentation:

Parameters:
- **decim** –
- **vlen** –

integrate_ii_sptr.**active_thread_priority**(*integrate_ii_sptr self*) → int

integrate_ii_sptr.**declare_sample_delay**(*integrate_ii_sptr self*, *int which*, *int delay*)
declare_sample_delay(integrate_ii_sptr self, unsigned int delay)

integrate_ii_sptr.**message_subscribers**(*integrate_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

integrate_ii_sptr.**min_noutput_items**(*integrate_ii_sptr self*) → int

integrate_ii_sptr.**pc_input_buffers_full_avg**(*integrate_ii_sptr self*, *int which*) → float
pc_input_buffers_full_avg(integrate_ii_sptr self) -> pmt_vector_float

integrate_ii_sptr.**pc_noutput_items_avg**(*integrate_ii_sptr self*) → float

integrate_ii_sptr.**pc_nproduced_avg**(*integrate_ii_sptr self*) → float

integrate_ii_sptr.**pc_output_buffers_full_avg**(*integrate_ii_sptr self*, *int which*) → float
pc_output_buffers_full_avg(integrate_ii_sptr self) -> pmt_vector_float

integrate_ii_sptr.**pc_throughput_avg**(*integrate_ii_sptr self*) → float

integrate_ii_sptr.**pc_work_time_avg**(*integrate_ii_sptr self*) → float

integrate_ii_sptr.**pc_work_time_total**(*integrate_ii_sptr self*) → float

integrate_ii_sptr.**sample_delay**(*integrate_ii_sptr self*, *int which*) → unsigned int

integrate_ii_sptr.**set_min_noutput_items**(*integrate_ii_sptr self*, *int m*)

integrate_ii_sptr.**set_thread_priority**(*integrate_ii_sptr self*, *int priority*) → int

integrate_ii_sptr.**thread_priority**(*integrate_ii_sptr self*) → int

gnuradio.blocks.**integrate_ss**(*int decim*, *int vlen=1*) → integrate_ss_sptr
Integrate successive samples and decimate.

Constructor Specific Documentation:

Parameters:
- **decim** –
- **vlen** –

integrate_ss_sptr.**active_thread_priority**(*integrate_ss_sptr self*) → int

integrate_ss_sptr.**declare_sample_delay**(*integrate_ss_sptr self*, *int which*, *int delay*)
declare_sample_delay(integrate_ss_sptr self, unsigned int delay)

integrate_ss_sptr.**message_subscribers**(*integrate_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

integrate_ss_sptr.**min_noutput_items**(*integrate_ss_sptr self*) → int

integrate_ss_sptr.**pc_input_buffers_full_avg**(*integrate_ss_sptr self*, *int which*) → float
pc_input_buffers_full_avg(integrate_ss_sptr self) -> pmt_vector_float

integrate_ss_sptr.**pc_noutput_items_avg**(*integrate_ss_sptr self*) → float

integrate_ss_sptr.**pc_nproduced_avg**(*integrate_ss_sptr self*) → float

integrate_ss_sptr.**pc_output_buffers_full_avg**(*integrate_ss_sptr self*, *int which*) → float
pc_output_buffers_full_avg(integrate_ss_sptr self) -> pmt_vector_float

integrate_ss_sptr.**pc_throughput_avg**(*integrate_ss_sptr self*) → float

integrate_ss_sptr.**pc_work_time_avg**(*integrate_ss_sptr self*) → float

integrate_ss_sptr.**pc_work_time_total**(*integrate_ss_sptr self*) → float

integrate_ss_sptr.**sample_delay**(*integrate_ss_sptr self*, *int which*) → unsigned int

integrate_ss_sptr.**set_min_noutput_items**(*integrate_ss_sptr self*, *int m*)

integrate_ss_sptr.**set_thread_priority**(*integrate_ss_sptr self*, *int priority*) → int

integrate_ss_sptr.**thread_priority**(*integrate_ss_sptr self*) → int

gnuradio.blocks.**interleave**(*size_t itemsize*, *unsigned int blocksize=1*) → interleave_sptr
interleave N inputs into a single output

This block interleaves blocks of samples. For each input connection, the samples are interleaved successively to the output connection. By default, the block interleaves a single sample from each input to the output unless blocksize is given in the constructor.

Constructor Specific Documentation:

Make a stream interleave block.

Parameters:
- **itemsize** – stream itemsize
- **blocksize** – size of block of samples to interleave

interleave_sptr.**active_thread_priority**(*interleave_sptr self*) → int

interleave_sptr.**declare_sample_delay**(*interleave_sptr self*, *int which*, *int delay*)
declare_sample_delay(interleave_sptr self, unsigned int delay)

interleave_sptr.**message_subscribers**(*interleave_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

interleave_sptr.**min_noutput_items**(*interleave_sptr self*) → int

interleave_sptr.**pc_input_buffers_full_avg**(*interleave_sptr self*, *int which*) → float
pc_input_buffers_full_avg(interleave_sptr self) -> pmt_vector_float

interleave_sptr.**pc_noutput_items_avg**(*interleave_sptr self*) → float

interleave_sptr.**pc_nproduced_avg**(*interleave_sptr self*) → float

interleave_sptr.**pc_output_buffers_full_avg**(*interleave_sptr self*, *int which*) → float
pc_output_buffers_full_avg(interleave_sptr self) -> pmt_vector_float

interleave_sptr.**pc_throughput_avg**(*interleave_sptr self*) → float

interleave_sptr.**pc_work_time_avg**(*interleave_sptr self*) → float

interleave_sptr.**pc_work_time_total**(*interleave_sptr self*) → float

interleave_sptr.**sample_delay**(*interleave_sptr self*, *int which*) → unsigned int

interleave_sptr.**set_min_noutput_items**(*interleave_sptr self*, *int m*)

interleave_sptr.**set_thread_priority**(*interleave_sptr self*, *int priority*) → int

interleave_sptr.**thread_priority**(*interleave_sptr self*) → int

gnuradio.blocks.**interleaved_char_to_complex**(*bool vector_input=False*) → interleaved_char_to_complex_sptr

Convert stream of interleaved chars to a stream of complex.

Constructor Specific Documentation:

Build an interleaved char to complex block.

Parameters: **vector_input** –

interleaved_char_to_complex_sptr.**active_thread_priority**(*interleaved_char_to_complex_sptr self*) → int

interleaved_char_to_complex_sptr.**declare_sample_delay**(*interleaved_char_to_complex_sptr self*, *int which*, *int delay*)

declare_sample_delay(interleaved_char_to_complex_sptr self, unsigned int delay)

interleaved_char_to_complex_sptr.**message_subscribers**(*interleaved_char_to_complex_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

interleaved_char_to_complex_sptr.**min_noutput_items**(*interleaved_char_to_complex_sptr self*) → int

interleaved_char_to_complex_sptr.**pc_input_buffers_full_avg**(*interleaved_char_to_complex_sptr self*, *int which*) → float

pc_input_buffers_full_avg(interleaved_char_to_complex_sptr self) -> pmt_vector_float

interleaved_char_to_complex_sptr.**pc_noutput_items_avg**(*interleaved_char_to_complex_sptr self*) → float

interleaved_char_to_complex_sptr.**pc_nproduced_avg**(*interleaved_char_to_complex_sptr self*) → float

interleaved_char_to_complex_sptr.**pc_output_buffers_full_avg**(*interleaved_char_to_complex_sptr self*, *int which*) → float

pc_output_buffers_full_avg(interleaved_char_to_complex_sptr self) -> pmt_vector_float

interleaved_char_to_complex_sptr.**pc_throughput_avg**(*interleaved_char_to_complex_sptr self*) → float

interleaved_char_to_complex_sptr.**pc_work_time_avg**(*interleaved_char_to_complex_sptr self*) → float

interleaved_char_to_complex_sptr.**pc_work_time_total**(*interleaved_char_to_complex_sptr self*) → float

interleaved_char_to_complex_sptr.**sample_delay**(*interleaved_char_to_complex_sptr self*, *int which*) → unsigned int

interleaved_char_to_complex_sptr.**set_min_noutput_items**(*interleaved_char_to_complex_sptr self*, *int m*)

interleaved_char_to_complex_sptr.**set_thread_priority**(*interleaved_char_to_complex_sptr self*, *int priority*) → int

interleaved_char_to_complex_sptr.**thread_priority**(*interleaved_char_to_complex_sptr self*) → int

gnuradio.blocks.**interleaved_short_to_complex**(*bool vector_input=False*, *bool swap=False*) → interleaved_short_to_complex_sptr

Convert stream of interleaved shorts to a stream of complex.

Constructor Specific Documentation:

Build an interleaved short to complex block.

Parameters:
- **vector_input** –
- **swap** –

interleaved_short_to_complex_sptr.**active_thread_priority**(*interleaved_short_to_complex_sptr self*) → int

interleaved_short_to_complex_sptr.**declare_sample_delay**(*interleaved_short_to_complex_sptr self*, *int*

which, *int delay*)

declare_sample_delay(interleaved_short_to_complex_sptr self, unsigned int delay)

interleaved_short_to_complex_sptr.**message_subscribers**(*interleaved_short_to_complex_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

interleaved_short_to_complex_sptr.**min_noutput_items**(*interleaved_short_to_complex_sptr self*) → int

interleaved_short_to_complex_sptr.**pc_input_buffers_full_avg**(*interleaved_short_to_complex_sptr self*, *int which*) → float

pc_input_buffers_full_avg(interleaved_short_to_complex_sptr self) -> pmt_vector_float

interleaved_short_to_complex_sptr.**pc_noutput_items_avg**(*interleaved_short_to_complex_sptr self*) → float

interleaved_short_to_complex_sptr.**pc_nproduced_avg**(*interleaved_short_to_complex_sptr self*) → float

interleaved_short_to_complex_sptr.**pc_output_buffers_full_avg**(*interleaved_short_to_complex_sptr self*, *int which*) → float

pc_output_buffers_full_avg(interleaved_short_to_complex_sptr self) -> pmt_vector_float

interleaved_short_to_complex_sptr.**pc_throughput_avg**(*interleaved_short_to_complex_sptr self*) → float

interleaved_short_to_complex_sptr.**pc_work_time_avg**(*interleaved_short_to_complex_sptr self*) → float

interleaved_short_to_complex_sptr.**pc_work_time_total**(*interleaved_short_to_complex_sptr self*) → float

interleaved_short_to_complex_sptr.**sample_delay**(*interleaved_short_to_complex_sptr self*, *int which*) → unsigned int

interleaved_short_to_complex_sptr.**set_min_noutput_items**(*interleaved_short_to_complex_sptr self*, *int m*)

interleaved_short_to_complex_sptr.**set_swap**(*interleaved_short_to_complex_sptr self*, *bool swap*)

interleaved_short_to_complex_sptr.**set_thread_priority**(*interleaved_short_to_complex_sptr self*, *int priority*) → int

interleaved_short_to_complex_sptr.**thread_priority**(*interleaved_short_to_complex_sptr self*) → int

gnuradio.blocks.**keep_m_in_n**(*size_t itemsize*, *int m*, *int n*, *int offset*) → keep_m_in_n_sptr

decimate a stream, keeping the first items out of every starting after items.

Constructor Specific Documentation:

Make a keep m in n block.

Parameters:
- **itemsize** – stream itemsize
- **m** – number of items to take in block of items
- **n** – block size in items
- **offset** – initial item offset into the stream

keep_m_in_n_sptr.**active_thread_priority**(*keep_m_in_n_sptr self*) → int

keep_m_in_n_sptr.**declare_sample_delay**(*keep_m_in_n_sptr self*, *int which*, *int delay*)

declare_sample_delay(keep_m_in_n_sptr self, unsigned int delay)

keep_m_in_n_sptr.**message_subscribers**(*keep_m_in_n_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

keep_m_in_n_sptr.**min_noutput_items**(*keep_m_in_n_sptr self*) → int

keep_m_in_n_sptr.**pc_input_buffers_full_avg**(*keep_m_in_n_sptr self*, *int which*) → float

pc_input_buffers_full_avg(keep_m_in_n_sptr self) -> pmt_vector_float

keep_m_in_n_sptr.**pc_noutput_items_avg**(*keep_m_in_n_sptr self*) → float

keep_m_in_n_sptr.**pc_nproduced_avg**(*keep_m_in_n_sptr self*) → float

keep_m_in_n_sptr.**pc_output_buffers_full_avg**(*keep_m_in_n_sptr self*, *int which*) → float

pc_output_buffers_full_avg(keep_m_in_n_sptr self) -> pmt_vector_float

keep_m_in_n_sptr.**pc_throughput_avg**(*keep_m_in_n_sptr self*) → float

keep_m_in_n_sptr.**pc_work_time_avg**(*keep_m_in_n_sptr self*) → float

keep_m_in_n_sptr.**pc_work_time_total**(*keep_m_in_n_sptr self*) → float

keep_m_in_n_sptr.**sample_delay**(*keep_m_in_n_sptr self*, *int which*) → unsigned int

keep_m_in_n_sptr.**set_m**(*keep_m_in_n_sptr self*, *int m*)

keep_m_in_n_sptr.**set_min_noutput_items**(*keep_m_in_n_sptr self*, *int m*)

keep_m_in_n_sptr.**set_n**(*keep_m_in_n_sptr self*, *int n*)

keep_m_in_n_sptr.**set_offset**(*keep_m_in_n_sptr self*, *int offset*)

keep_m_in_n_sptr.**set_thread_priority**(*keep_m_in_n_sptr self*, *int priority*) → int

keep_m_in_n_sptr.**thread_priority**(*keep_m_in_n_sptr self*) → int

gnuradio.blocks.**keep_one_in_n**(*size_t itemsize*, *int n*) → keep_one_in_n_sptr

decimate a stream, keeping the last item out of every .

Constructor Specific Documentation:

Make a keep one in n block.

Parameters:
- **itemsize** – stream itemsize
- **n** – block size in items

keep_one_in_n_sptr.**active_thread_priority**(*keep_one_in_n_sptr self*) → int

keep_one_in_n_sptr.**declare_sample_delay**(*keep_one_in_n_sptr self*, *int which*, *int delay*)

declare_sample_delay(keep_one_in_n_sptr self, unsigned int delay)

keep_one_in_n_sptr.**message_subscribers**(*keep_one_in_n_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

keep_one_in_n_sptr.**min_noutput_items**(*keep_one_in_n_sptr self*) → int

keep_one_in_n_sptr.**pc_input_buffers_full_avg**(*keep_one_in_n_sptr self*, *int which*) → float

pc_input_buffers_full_avg(keep_one_in_n_sptr self) -> pmt_vector_float

keep_one_in_n_sptr.**pc_noutput_items_avg**(*keep_one_in_n_sptr self*) → float

keep_one_in_n_sptr.**pc_nproduced_avg**(*keep_one_in_n_sptr self*) → float

keep_one_in_n_sptr.**pc_output_buffers_full_avg**(*keep_one_in_n_sptr self*, *int which*) → float

pc_output_buffers_full_avg(keep_one_in_n_sptr self) -> pmt_vector_float

keep_one_in_n_sptr.**pc_throughput_avg**(*keep_one_in_n_sptr self*) → float

keep_one_in_n_sptr.**pc_work_time_avg**(*keep_one_in_n_sptr self*) → float

keep_one_in_n_sptr.**pc_work_time_total**(*keep_one_in_n_sptr self*) → float

keep_one_in_n_sptr.**sample_delay**(*keep_one_in_n_sptr self*, *int which*) → unsigned int

keep_one_in_n_sptr.**set_min_noutput_items**(*keep_one_in_n_sptr self*, *int m*)

keep_one_in_n_sptr.**set_n**(*keep_one_in_n_sptr self*, *int n*)

keep_one_in_n_sptr.**set_thread_priority**(*keep_one_in_n_sptr self*, *int priority*) → int

keep_one_in_n_sptr.**thread_priority**(*keep_one_in_n_sptr self*) → int

gnuradio.blocks.**lfsr_32k_source_s**() → lfsr_32k_source_s_sptr

LFSR pseudo-random source with period of 2^15 bits (2^11 shorts)

This source is typically used along with gr::blocks::check_lfsr_32k_s to test the USRP using its digital loopback mode.

Constructor Specific Documentation:

Make a LFSR 32k source block.

lfsr_32k_source_s_sptr.**active_thread_priority**(*lfsr_32k_source_s_sptr self*) → int

lfsr_32k_source_s_sptr.**declare_sample_delay**(*lfsr_32k_source_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(lfsr_32k_source_s_sptr self, unsigned int delay)

lfsr_32k_source_s_sptr.**message_subscribers**(*lfsr_32k_source_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

lfsr_32k_source_s_sptr.**min_noutput_items**(*lfsr_32k_source_s_sptr self*) → int

lfsr_32k_source_s_sptr.**pc_input_buffers_full_avg**(*lfsr_32k_source_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(lfsr_32k_source_s_sptr self) -> pmt_vector_float

lfsr_32k_source_s_sptr.**pc_noutput_items_avg**(*lfsr_32k_source_s_sptr self*) → float

lfsr_32k_source_s_sptr.**pc_nproduced_avg**(*lfsr_32k_source_s_sptr self*) → float

lfsr_32k_source_s_sptr.**pc_output_buffers_full_avg**(*lfsr_32k_source_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(lfsr_32k_source_s_sptr self) -> pmt_vector_float

lfsr_32k_source_s_sptr.**pc_throughput_avg**(*lfsr_32k_source_s_sptr self*) → float

lfsr_32k_source_s_sptr.**pc_work_time_avg**(*lfsr_32k_source_s_sptr self*) → float

lfsr_32k_source_s_sptr.**pc_work_time_total**(*lfsr_32k_source_s_sptr self*) → float

lfsr_32k_source_s_sptr.**sample_delay**(*lfsr_32k_source_s_sptr self*, *int which*) → unsigned int

lfsr_32k_source_s_sptr.**set_min_noutput_items**(*lfsr_32k_source_s_sptr self*, *int m*)

lfsr_32k_source_s_sptr.**set_thread_priority**(*lfsr_32k_source_s_sptr self*, *int priority*) → int

lfsr_32k_source_s_sptr.**thread_priority**(*lfsr_32k_source_s_sptr self*) → int

gnuradio.blocks.**max_ff**(*size_t vlen*, *size_t vlen_out=1*) → max_ff_sptr

Compares vectors from multiple streams and determines the maximum value from each vector over all streams.

Data is passed in as a vector of length from multiple input sources. If vlen_out == 1 then It will look through these streams of data items and the output stream will contain the maximum value in the vector. If vlen_out == vlen and not equal to 1 then output will be a vector with individual items selected from the maximum corresponding input vector items.

Constructor Specific Documentation:

Parameters:
- **vlen** –
- **vlen_out** –

max_ff_sptr.**active_thread_priority**(*max_ff_sptr self*) → int

max_ff_sptr.**declare_sample_delay**(*max_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(max_ff_sptr self, unsigned int delay)

max_ff_sptr.**message_subscribers**(*max_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

max_ff_sptr.**min_noutput_items**(*max_ff_sptr self*) → int

max_ff_sptr.**pc_input_buffers_full_avg**(*max_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(max_ff_sptr self) -> pmt_vector_float

max_ff_sptr.**pc_noutput_items_avg**(*max_ff_sptr self*) → float

max_ff_sptr.**pc_nproduced_avg**(*max_ff_sptr self*) → float

max_ff_sptr.**pc_output_buffers_full_avg**(*max_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(max_ff_sptr self) -> pmt_vector_float

max_ff_sptr.**pc_throughput_avg**(*max_ff_sptr self*) → float

max_ff_sptr.**pc_work_time_avg**(*max_ff_sptr self*) → float

max_ff_sptr.**pc_work_time_total**(*max_ff_sptr self*) → float

max_ff_sptr.**sample_delay**(*max_ff_sptr self*, *int which*) → unsigned int

max_ff_sptr.**set_min_noutput_items**(*max_ff_sptr self*, *int m*)

max_ff_sptr.**set_thread_priority**(*max_ff_sptr self*, *int priority*) → int

max_ff_sptr.**thread_priority**(*max_ff_sptr self*) → int

gnuradio.blocks.**max_ii**(*size_t vlen*, *size_t vlen_out=1*) → max_ii_sptr

Compares vectors from multiple streams and determines the maximum value from each vector over all streams.

Data is passed in as a vector of length from multiple input sources. If vlen_out == 1 then It will look through these streams of data items and the output stream will contain the maximum value in the vector. If vlen_out == vlen and not equal to 1 then output will be a vector with individual items selected from the maximum corresponding input vector items.

Constructor Specific Documentation:

Parameters:
- **vlen** –
- **vlen_out** –

max_ii_sptr.**active_thread_priority**(*max_ii_sptr self*) → int

max_ii_sptr.**declare_sample_delay**(*max_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(max_ii_sptr self, unsigned int delay)

max_ii_sptr.**message_subscribers**(*max_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

max_ii_sptr.**min_noutput_items**(*max_ii_sptr self*) → int

max_ii_sptr.**pc_input_buffers_full_avg**(*max_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(max_ii_sptr self) -> pmt_vector_float

max_ii_sptr.**pc_noutput_items_avg**(*max_ii_sptr self*) → float

max_ii_sptr.**pc_nproduced_avg**(*max_ii_sptr self*) → float

max_ii_sptr.**pc_output_buffers_full_avg**(*max_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(max_ii_sptr self) -> pmt_vector_float

max_ii_sptr.**pc_throughput_avg**(*max_ii_sptr self*) → float

max_ii_sptr.**pc_work_time_avg**(*max_ii_sptr self*) → float

max_ii_sptr.**pc_work_time_total**(*max_ii_sptr self*) → float

max_ii_sptr.**sample_delay**(*max_ii_sptr self*, *int which*) → unsigned int

max_ii_sptr.**set_min_noutput_items**(*max_ii_sptr self*, *int m*)

max_ii_sptr.**set_thread_priority**(*max_ii_sptr self*, *int priority*) → int

max_ii_sptr.**thread_priority**(*max_ii_sptr self*) → int

gnuradio.blocks.**max_ss**(*size_t vlen*, *size_t vlen_out=1*) → max_ss_sptr

Compares vectors from multiple streams and determines the maximum value from each vector over all streams.

Data is passed in as a vector of length from multiple input sources. If vlen_out == 1 then It will look through these streams of data items and the output stream will contain the maximum value in the vector. If vlen_out == vlen and not equal to 1 then output will be a vector with individual items selected from the maximum corresponding input vector items.

Constructor Specific Documentation:

Parameters:
- **vlen** –
- **vlen_out** –

max_ss_sptr.**active_thread_priority**(*max_ss_sptr self*) → int

max_ss_sptr.**declare_sample_delay**(*max_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(max_ss_sptr self, unsigned int delay)

max_ss_sptr.**message_subscribers**(*max_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

max_ss_sptr.**min_noutput_items**(*max_ss_sptr self*) → int

max_ss_sptr.**pc_input_buffers_full_avg**(*max_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(max_ss_sptr self) -> pmt_vector_float

max_ss_sptr.**pc_noutput_items_avg**(*max_ss_sptr self*) → float

max_ss_sptr.**pc_nproduced_avg**(*max_ss_sptr self*) → float

max_ss_sptr.**pc_output_buffers_full_avg**(*max_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(max_ss_sptr self) -> pmt_vector_float

max_ss_sptr.**pc_throughput_avg**(*max_ss_sptr self*) → float

max_ss_sptr.**pc_work_time_avg**(*max_ss_sptr self*) → float

max_ss_sptr.**pc_work_time_total**(*max_ss_sptr self*) → float

max_ss_sptr.**sample_delay**(*max_ss_sptr self*, *int which*) → unsigned int

max_ss_sptr.**set_min_noutput_items**(*max_ss_sptr self*, *int m*)

max_ss_sptr.**set_thread_priority**(*max_ss_sptr self*, *int priority*) → int

max_ss_sptr.**thread_priority**(*max_ss_sptr self*) → int

gnuradio.blocks.**message_burst_source**(*size_t itemsize*, *int msgq_limit*) → message_burst_source_sptr

make(size_t itemsize, msg_queue_sptr msgq) -> message_burst_source_sptr

Turn received messages into a stream and tag them for UHD to send.

Constructor Specific Documentation:

Parameters:
- **itemsize** –
- **msgq_limit** –

message_burst_source_sptr.**active_thread_priority**(*message_burst_source_sptr self*) → int

message_burst_source_sptr.**declare_sample_delay**(*message_burst_source_sptr self*, *int which*, *int delay*)

declare_sample_delay(message_burst_source_sptr self, unsigned int delay)

message_burst_source_sptr.**message_subscribers**(*message_burst_source_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

message_burst_source_sptr.**min_noutput_items**(*message_burst_source_sptr self*) → int

message_burst_source_sptr.**msgq**(*message_burst_source_sptr self*) → msg_queue_sptr

message_burst_source_sptr.**pc_input_buffers_full_avg**(*message_burst_source_sptr self*, *int which*) → float

pc_input_buffers_full_avg(message_burst_source_sptr self) -> pmt_vector_float

message_burst_source_sptr.**pc_noutput_items_avg**(*message_burst_source_sptr self*) → float

message_burst_source_sptr.**pc_nproduced_avg**(*message_burst_source_sptr self*) → float

message_burst_source_sptr.**pc_output_buffers_full_avg**(*message_burst_source_sptr self*, *int which*) → float

pc_output_buffers_full_avg(message_burst_source_sptr self) -> pmt_vector_float

message_burst_source_sptr.**pc_throughput_avg**(*message_burst_source_sptr self*) → float

message_burst_source_sptr.**pc_work_time_avg**(*message_burst_source_sptr self*) → float

message_burst_source_sptr.**pc_work_time_total**(*message_burst_source_sptr self*) → float

message_burst_source_sptr.**sample_delay**(*message_burst_source_sptr self*, *int which*) → unsigned int

message_burst_source_sptr.**set_min_noutput_items**(*message_burst_source_sptr self*, *int m*)

message_burst_source_sptr.**set_thread_priority**(*message_burst_source_sptr self*, *int priority*) → int

message_burst_source_sptr.**thread_priority**(*message_burst_source_sptr self*) → int

gnuradio.blocks.**message_debug**() → message_debug_sptr

Debug block for the message passing system.

The message debug block is used to capture and print or store messages as they are received. Any block that generates a message may connect that message port to one or more of the three message input ports of this debug block. The message ports are:

Constructor Specific Documentation:

Build the message debug block. It takes no parameters and has three message ports: print, store, and print_pdu.

message_debug_sptr.**active_thread_priority**(*message_debug_sptr self*) → int

message_debug_sptr.**declare_sample_delay**(*message_debug_sptr self*, *int which*, *int delay*)

declare_sample_delay(message_debug_sptr self, unsigned int delay)

message_debug_sptr.**get_message**(*message_debug_sptr self*, *int i*) → swig_int_ptr

Get a message (as a PMT) from the message vector at index .

Messages passed to the 'store' port will be stored in a vector. This function retrieves those messages by index. They are index in order of when they were received (all messages are just pushed onto the back of a vector). This is mostly useful in debugging message passing graphs and in QA code.

message_debug_sptr.**message_subscribers**(*message_debug_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

message_debug_sptr.**min_noutput_items**(*message_debug_sptr self*) → int

message_debug_sptr.**num_messages**(*message_debug_sptr self*) → int

Reports the number of messages received by this block.

message_debug_sptr.**pc_input_buffers_full_avg**(*message_debug_sptr self*, *int which*) → float

pc_input_buffers_full_avg(message_debug_sptr self) -> pmt_vector_float

message_debug_sptr.**pc_noutput_items_avg**(*message_debug_sptr self*) → float

message_debug_sptr.**pc_nproduced_avg**(*message_debug_sptr self*) → float

message_debug_sptr.**pc_output_buffers_full_avg**(*message_debug_sptr self*, *int which*) → float

pc_output_buffers_full_avg(message_debug_sptr self) -> pmt_vector_float

message_debug_sptr.**pc_throughput_avg**(*message_debug_sptr self*) → float

message_debug_sptr.**pc_work_time_avg**(*message_debug_sptr self*) → float

message_debug_sptr.**pc_work_time_total**(*message_debug_sptr self*) → float

message_debug_sptr.**sample_delay**(*message_debug_sptr self*, *int which*) → unsigned int

message_debug_sptr.**set_min_noutput_items**(*message_debug_sptr self*, *int m*)

message_debug_sptr.**set_thread_priority**(*message_debug_sptr self*, *int priority*) → int

message_debug_sptr.**thread_priority**(*message_debug_sptr self*) → int

gnuradio.blocks.**message_sink**(*size_t itemsize*, *msg_queue_sptr msgq*, *bool dont_block*) → message_sink_sptr

make(size_t itemsize, msg_queue_sptr msgq, bool dont_block, std::string const & lengthtagname) -> message_sink_sptr

Gather received items into messages and insert into msgq.

Constructor Specific Documentation:

Parameters:
- **itemsize** –
- **msgq** –
- **dont_block** –

message_sink_sptr.**active_thread_priority**(*message_sink_sptr self*) → int

message_sink_sptr.**declare_sample_delay**(*message_sink_sptr self*, *int which*, *int delay*)

declare_sample_delay(message_sink_sptr self, unsigned int delay)

message_sink_sptr.**message_subscribers**(*message_sink_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

message_sink_sptr.**min_noutput_items**(*message_sink_sptr self*) → int

message_sink_sptr.**pc_input_buffers_full_avg**(*message_sink_sptr self*, *int which*) → float

pc_input_buffers_full_avg(message_sink_sptr self) -> pmt_vector_float

message_sink_sptr.**pc_noutput_items_avg**(*message_sink_sptr self*) → float

message_sink_sptr.**pc_nproduced_avg**(*message_sink_sptr self*) → float

message_sink_sptr.**pc_output_buffers_full_avg**(*message_sink_sptr self*, *int which*) → float

pc_output_buffers_full_avg(message_sink_sptr self) -> pmt_vector_float

message_sink_sptr.**pc_throughput_avg**(*message_sink_sptr self*) → float

message_sink_sptr.**pc_work_time_avg**(*message_sink_sptr self*) → float

message_sink_sptr.**pc_work_time_total**(*message_sink_sptr self*) → float

message_sink_sptr.**sample_delay**(*message_sink_sptr self*, *int which*) → unsigned int

message_sink_sptr.**set_min_noutput_items**(*message_sink_sptr self*, *int m*)

message_sink_sptr.**set_thread_priority**(*message_sink_sptr self*, *int priority*) → int

message_sink_sptr.**thread_priority**(*message_sink_sptr self*) → int

gnuradio.blocks.**message_source**(*size_t itemsize*, *int msgq_limit=0*) → message_source_sptr

make(size_t itemsize, msg_queue_sptr msgq) -> message_source_sptr make(size_t itemsize, msg_queue_sptr msgq, std::string const & lengthtagname) -> message_source_sptr

Turn received messages into a stream.

Constructor Specific Documentation:

Parameters:
- **itemsize** –
- **msgq_limit** –

message_source_sptr.**active_thread_priority**(*message_source_sptr self*) → int

message_source_sptr.**declare_sample_delay**(*message_source_sptr self*, *int which*, *int delay*)

declare_sample_delay(message_source_sptr self, unsigned int delay)

message_source_sptr.**message_subscribers**(*message_source_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

message_source_sptr.**min_noutput_items**(*message_source_sptr self*) → int

message_source_sptr.**msgq**(*message_source_sptr self*) → msg_queue_sptr

message_source_sptr.**pc_input_buffers_full_avg**(*message_source_sptr self*, *int which*) → float

pc_input_buffers_full_avg(message_source_sptr self) -> pmt_vector_float

message_source_sptr.**pc_noutput_items_avg**(*message_source_sptr self*) → float

message_source_sptr.**pc_nproduced_avg**(*message_source_sptr self*) → float

message_source_sptr.**pc_output_buffers_full_avg**(*message_source_sptr self*, *int which*) → float

pc_output_buffers_full_avg(message_source_sptr self) -> pmt_vector_float

message_source_sptr.**pc_throughput_avg**(*message_source_sptr self*) → float

message_source_sptr.**pc_work_time_avg**(*message_source_sptr self*) → float

message_source_sptr.**pc_work_time_total**(*message_source_sptr self*) → float

message_source_sptr.**sample_delay**(*message_source_sptr self*, *int which*) → unsigned int

message_source_sptr.**set_min_noutput_items**(*message_source_sptr self*, *int m*)

message_source_sptr.**set_thread_priority**(*message_source_sptr self*, *int priority*) → int

message_source_sptr.**thread_priority**(*message_source_sptr self*) → int

gnuradio.blocks.**message_strobe**(*swig_int_ptr msg*, *float period_ms*) → message_strobe_sptr

Send message at defined interval.

Takes a PMT message and sends it out every milliseconds. Useful for testing/debugging the message system.

Constructor Specific Documentation:

Make a message stobe block to send message every milliseconds.

Parameters:
- **msg** – The message to send as a PMT.
- **period_ms** – the time period in milliseconds in which to send .

message_strobe_sptr.**active_thread_priority**(*message_strobe_sptr self*) → int

message_strobe_sptr.**declare_sample_delay**(*message_strobe_sptr self*, *int which*, *int delay*)

declare_sample_delay(message_strobe_sptr self, unsigned int delay)

message_strobe_sptr.**message_subscribers**(*message_strobe_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

message_strobe_sptr.**min_noutput_items**(*message_strobe_sptr self*) → int

message_strobe_sptr.**msg**(*message_strobe_sptr self*) → swig_int_ptr

Get the value of the message being sent.

message_strobe_sptr.**pc_input_buffers_full_avg**(*message_strobe_sptr self*, *int which*) → float

pc_input_buffers_full_avg(message_strobe_sptr self) -> pmt_vector_float

message_strobe_sptr.**pc_noutput_items_avg**(*message_strobe_sptr self*) → float

message_strobe_sptr.**pc_nproduced_avg**(*message_strobe_sptr self*) → float

message_strobe_sptr.**pc_output_buffers_full_avg**(*message_strobe_sptr self*, *int which*) → float

pc_output_buffers_full_avg(message_strobe_sptr self) -> pmt_vector_float

message_strobe_sptr.**pc_throughput_avg**(*message_strobe_sptr self*) → float

message_strobe_sptr.**pc_work_time_avg**(*message_strobe_sptr self*) → float

message_strobe_sptr.**pc_work_time_total**(*message_strobe_sptr self*) → float

message_strobe_sptr.**period**(*message_strobe_sptr self*) → float

Get the time interval of the strobe.

message_strobe_sptr.**sample_delay**(*message_strobe_sptr self*, *int which*) → unsigned int

message_strobe_sptr.**set_min_noutput_items**(*message_strobe_sptr self*, *int m*)

message_strobe_sptr.**set_msg**(*message_strobe_sptr self*, *swig_int_ptr msg*)

Reset the message being sent.

message_strobe_sptr.**set_period**(*message_strobe_sptr self*, *float period_ms*)

Reset the sending interval.

message_strobe_sptr.**set_thread_priority**(*message_strobe_sptr self*, *int priority*) → int

message_strobe_sptr.**thread_priority**(*message_strobe_sptr self*) → int

gnuradio.blocks.**message_strobe_random**(*swig_int_ptr msg*, *gr::blocks::message_strobe_random_distribution_t dist*, *float mean_ms*, *float std_ms*) → message_strobe_random_sptr

Send message at defined interval.

Takes a PMT message and sends it out every at random intervals. The interval is basedon a random distribution, , with specified mean () and variance (). Useful for testing/debugging the message system.

Constructor Specific Documentation:

Make a message stobe block to sends message at random intervals defined by the distribution with mean and standard deviation .

Parameters:
- **msg** – The message to send as a PMT.
- **dist** – The random distribution from which to draw events.
- **mean_ms** – The mean of the distribution.
- **std_ms** – The standard deviation of the distribution.

message_strobe_random_sptr.**active_thread_priority**(*message_strobe_random_sptr self*) → int

message_strobe_random_sptr.**declare_sample_delay**(*message_strobe_random_sptr self*, *int which*, *int delay*)

declare_sample_delay(message_strobe_random_sptr self, unsigned int delay)

message_strobe_random_sptr.**dist**(*message_strobe_random_sptr self*) → gr::blocks::message_strobe_random_distribution_t

get the current distribution.

message_strobe_random_sptr.**mean**(*message_strobe_random_sptr self*) → float

Get the time interval of the strobe_random.

message_strobe_random_sptr.**message_subscribers**(*message_strobe_random_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

message_strobe_random_sptr.**min_noutput_items**(*message_strobe_random_sptr self*) → int

message_strobe_random_sptr.**msg**(*message_strobe_random_sptr self*) → swig_int_ptr

Get the value of the message being sent.

message_strobe_random_sptr.**pc_input_buffers_full_avg**(*message_strobe_random_sptr self*, *int which*) → float

pc_input_buffers_full_avg(message_strobe_random_sptr self) -> pmt_vector_float

message_strobe_random_sptr.**pc_noutput_items_avg**(*message_strobe_random_sptr self*) → float

message_strobe_random_sptr.**pc_nproduced_avg**(*message_strobe_random_sptr self*) → float

message_strobe_random_sptr.**pc_output_buffers_full_avg**(*message_strobe_random_sptr self*, *int which*) → float

pc_output_buffers_full_avg(message_strobe_random_sptr self) -> pmt_vector_float

message_strobe_random_sptr.**pc_throughput_avg**(*message_strobe_random_sptr self*) → float

message_strobe_random_sptr.**pc_work_time_avg**(*message_strobe_random_sptr self*) → float

message_strobe_random_sptr.**pc_work_time_total**(*message_strobe_random_sptr self*) → float

message_strobe_random_sptr.**sample_delay**(*message_strobe_random_sptr self*, *int which*) → unsigned int

message_strobe_random_sptr.**set_dist**(*message_strobe_random_sptr self*, *gr::blocks::message_strobe_random_distribution_t dist*)

message_strobe_random_sptr.**set_mean**(*message_strobe_random_sptr self*, *float mean*)

Reset the sending interval.

message_strobe_random_sptr.**set_min_noutput_items**(*message_strobe_random_sptr self*, *int m*)

message_strobe_random_sptr.**set_msg**(*message_strobe_random_sptr self*, *swig_int_ptr msg*)

Reset the message being sent.

message_strobe_random_sptr.**set_std**(*message_strobe_random_sptr self*, *float std*)

Reset the sending interval.

message_strobe_random_sptr.**set_thread_priority**(*message_strobe_random_sptr self*, *int priority*) → int

message_strobe_random_sptr.**std**(*message_strobe_random_sptr self*) → float

Get the std of strobe_random.

message_strobe_random_sptr.**thread_priority**(*message_strobe_random_sptr self*) → int

gnuradio.blocks.**min_ff**(*size_t vlen*, *size_t vlen_out=1*) → min_ff_sptr

Compares vectors from multiple streams and determines the minimum value from each vector over all streams.

Data is passed in as a vector of length from multiple input sources. If vlen_out == 1 then It will look through these streams of data items and the output stream will contain the minimum value in the vector. If vlen_out == vlen and not equal to 1 then output will be a vector with individual items selected from the minimum corresponding input vector items.

Constructor Specific Documentation:

Parameters:
- **vlen** –
- **vlen_out** –

min_ff_sptr.**active_thread_priority**(*min_ff_sptr self*) → int

min_ff_sptr.**declare_sample_delay**(*min_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(min_ff_sptr self, unsigned int delay)

min_ff_sptr.**message_subscribers**(*min_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

min_ff_sptr.**min_noutput_items**(*min_ff_sptr self*) → int

min_ff_sptr.**pc_input_buffers_full_avg**(*min_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(min_ff_sptr self) -> pmt_vector_float

min_ff_sptr.**pc_noutput_items_avg**(*min_ff_sptr self*) → float

min_ff_sptr.**pc_nproduced_avg**(*min_ff_sptr self*) → float

min_ff_sptr.**pc_output_buffers_full_avg**(*min_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(min_ff_sptr self) -> pmt_vector_float

min_ff_sptr.**pc_throughput_avg**(*min_ff_sptr self*) → float

min_ff_sptr.**pc_work_time_avg**(*min_ff_sptr self*) → float

min_ff_sptr.**pc_work_time_total**(*min_ff_sptr self*) → float

min_ff_sptr.**sample_delay**(*min_ff_sptr self*, *int which*) → unsigned int

min_ff_sptr.**set_min_noutput_items**(*min_ff_sptr self*, *int m*)

min_ff_sptr.**set_thread_priority**(*min_ff_sptr self*, *int priority*) → int

min_ff_sptr.**thread_priority**(*min_ff_sptr self*) → int

gnuradio.blocks.**min_ii**(*size_t vlen*, *size_t vlen_out=1*) → min_ii_sptr

Compares vectors from multiple streams and determines the minimum value from each vector over all streams.

Data is passed in as a vector of length from multiple input sources. If vlen_out == 1 then It will look through these streams of data items and the output stream will contain the minimum value in the vector. If vlen_out == vlen and not equal to 1 then output will be a vector with individual items selected from the minimum corresponding input vector items.

Constructor Specific Documentation:

Parameters:
- **vlen** –
- **vlen_out** –

min_ii_sptr.**active_thread_priority**(*min_ii_sptr self*) → int

min_ii_sptr.**declare_sample_delay**(*min_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(min_ii_sptr self, unsigned int delay)

min_ii_sptr.**message_subscribers**(*min_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

min_ii_sptr.**min_noutput_items**(*min_ii_sptr self*) → int

min_ii_sptr.**pc_input_buffers_full_avg**(*min_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(min_ii_sptr self) -> pmt_vector_float

min_ii_sptr.**pc_noutput_items_avg**(*min_ii_sptr self*) → float

min_ii_sptr.**pc_nproduced_avg**(*min_ii_sptr self*) → float

min_ii_sptr.**pc_output_buffers_full_avg**(*min_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(min_ii_sptr self) -> pmt_vector_float

min_ii_sptr.**pc_throughput_avg**(*min_ii_sptr self*) → float

min_ii_sptr.**pc_work_time_avg**(*min_ii_sptr self*) → float

min_ii_sptr.**pc_work_time_total**(*min_ii_sptr self*) → float

min_ii_sptr.**sample_delay**(*min_ii_sptr self*, *int which*) → unsigned int

min_ii_sptr.**set_min_noutput_items**(*min_ii_sptr self*, *int m*)

min_ii_sptr.**set_thread_priority**(*min_ii_sptr self*, *int priority*) → int

min_ii_sptr.**thread_priority**(*min_ii_sptr self*) → int

gnuradio.blocks.**min_ss**(*size_t vlen*, *size_t vlen_out=1*) → min_ss_sptr

Compares vectors from multiple streams and determines the minimum value from each vector over all streams.

Data is passed in as a vector of length from multiple input sources. If vlen_out == 1 then It will look through these streams of data items and the output stream will contain the minimum value in the vector. If vlen_out == vlen and not equal to 1 then output will be a vector with individual items selected from the minimum corresponding input vector items.

Constructor Specific Documentation:

Parameters:
- **vlen** –
- **vlen_out** –

min_ss_sptr.**active_thread_priority**(*min_ss_sptr self*) → int

min_ss_sptr.**declare_sample_delay**(*min_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(min_ss_sptr self, unsigned int delay)

min_ss_sptr.**message_subscribers**(*min_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

min_ss_sptr.**min_noutput_items**(*min_ss_sptr self*) → int

min_ss_sptr.**pc_input_buffers_full_avg**(*min_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(min_ss_sptr self) -> pmt_vector_float

min_ss_sptr.**pc_noutput_items_avg**(*min_ss_sptr self*) → float

min_ss_sptr.**pc_nproduced_avg**(*min_ss_sptr self*) → float

min_ss_sptr.**pc_output_buffers_full_avg**(*min_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(min_ss_sptr self) -> pmt_vector_float

min_ss_sptr.**pc_throughput_avg**(*min_ss_sptr self*) → float

min_ss_sptr.**pc_work_time_avg**(*min_ss_sptr self*) → float

min_ss_sptr.**pc_work_time_total**(*min_ss_sptr self*) → float

min_ss_sptr.**sample_delay**(*min_ss_sptr self*, *int which*) → unsigned int

min_ss_sptr.**set_min_noutput_items**(*min_ss_sptr self*, *int m*)

min_ss_sptr.**set_thread_priority**(*min_ss_sptr self*, *int priority*) → int

min_ss_sptr.**thread_priority**(*min_ss_sptr self*) → int

gnuradio.blocks.**moving_average_cc**(*int length*, *gr_complex scale*, *int max_iter=4096*, *unsigned int vlen=1*) → moving_average_cc_sptr

output is the moving sum of the last N samples, scaled by the scale factor

Constructor Specific Documentation:

Create a moving average block.

Parameters:
- **length** – Number of samples to use in the average.
- **scale** – scale factor for the result.
- **max_iter** – limits how long we go without flushing the accumulator This is necessary to avoid numerical instability for float and complex.
- **vlen** – When > 1, do a per-vector-element moving average

moving_average_cc_sptr.**active_thread_priority**(*moving_average_cc_sptr self*) → int

moving_average_cc_sptr.**declare_sample_delay**(*moving_average_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(moving_average_cc_sptr self, unsigned int delay)

moving_average_cc_sptr.**message_subscribers**(*moving_average_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

moving_average_cc_sptr.**min_noutput_items**(*moving_average_cc_sptr self*) → int

moving_average_cc_sptr.**pc_input_buffers_full_avg**(*moving_average_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(moving_average_cc_sptr self) -> pmt_vector_float

moving_average_cc_sptr.**pc_noutput_items_avg**(*moving_average_cc_sptr self*) → float

moving_average_cc_sptr.**pc_nproduced_avg**(*moving_average_cc_sptr self*) → float

moving_average_cc_sptr.**pc_output_buffers_full_avg**(*moving_average_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(moving_average_cc_sptr self) -> pmt_vector_float

moving_average_cc_sptr.**pc_throughput_avg**(*moving_average_cc_sptr self*) → float

moving_average_cc_sptr.**pc_work_time_avg**(*moving_average_cc_sptr self*) → float

moving_average_cc_sptr.**pc_work_time_total**(*moving_average_cc_sptr self*) → float

moving_average_cc_sptr.**sample_delay**(*moving_average_cc_sptr self*, *int which*) → unsigned int

moving_average_cc_sptr.**scale**(*moving_average_cc_sptr self*) → gr_complex

Get the scale factor being used.

moving_average_cc_sptr.**set_length**(*moving_average_cc_sptr self*, *int length*)

Set the length.

moving_average_cc_sptr.**set_length_and_scale**(*moving_average_cc_sptr self*, *int length*, *gr_complex scale*)

Set both the length and the scale factor together.

moving_average_cc_sptr.**set_min_noutput_items**(*moving_average_cc_sptr self*, *int m*)

moving_average_cc_sptr.**set_scale**(*moving_average_cc_sptr self*, *gr_complex scale*)

Set the scale factor.

moving_average_cc_sptr.**set_thread_priority**(*moving_average_cc_sptr self*, *int priority*) → int

moving_average_cc_sptr.**thread_priority**(*moving_average_cc_sptr self*) → int

gnuradio.blocks.**moving_average_ff**(*int length*, *float scale*, *int max_iter=4096*, *unsigned int vlen=1*) → moving_average_ff_sptr

output is the moving sum of the last N samples, scaled by the scale factor

Constructor Specific Documentation:

Create a moving average block.

Parameters:
- **length** – Number of samples to use in the average.
- **scale** – scale factor for the result.
- **max_iter** – limits how long we go without flushing the accumulator This is necessary to avoid numerical instability for float and complex.
- **vlen** – When > 1, do a per-vector-element moving average

moving_average_ff_sptr.**active_thread_priority**(*moving_average_ff_sptr self*) → int

moving_average_ff_sptr.**declare_sample_delay**(*moving_average_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(moving_average_ff_sptr self, unsigned int delay)

moving_average_ff_sptr.**message_subscribers**(*moving_average_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

moving_average_ff_sptr.**min_noutput_items**(*moving_average_ff_sptr self*) → int

moving_average_ff_sptr.**pc_input_buffers_full_avg**(*moving_average_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(moving_average_ff_sptr self) -> pmt_vector_float

moving_average_ff_sptr.**pc_noutput_items_avg**(*moving_average_ff_sptr self*) → float

moving_average_ff_sptr.**pc_nproduced_avg**(*moving_average_ff_sptr self*) → float

moving_average_ff_sptr.**pc_output_buffers_full_avg**(*moving_average_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(moving_average_ff_sptr self) -> pmt_vector_float

moving_average_ff_sptr.**pc_throughput_avg**(*moving_average_ff_sptr self*) → float

moving_average_ff_sptr.**pc_work_time_avg**(*moving_average_ff_sptr self*) → float

moving_average_ff_sptr.**pc_work_time_total**(*moving_average_ff_sptr self*) → float

moving_average_ff_sptr.**sample_delay**(*moving_average_ff_sptr self*, *int which*) → unsigned int

moving_average_ff_sptr.**scale**(*moving_average_ff_sptr self*) → float

Get the scale factor being used.

moving_average_ff_sptr.**set_length**(*moving_average_ff_sptr self*, *int length*)

Set the length.

moving_average_ff_sptr.**set_length_and_scale**(*moving_average_ff_sptr self*, *int length*, *float scale*)

Set both the length and the scale factor together.

moving_average_ff_sptr.**set_min_noutput_items**(*moving_average_ff_sptr self*, *int m*)

moving_average_ff_sptr.**set_scale**(*moving_average_ff_sptr self*, *float scale*)

Set the scale factor.

moving_average_ff_sptr.**set_thread_priority**(*moving_average_ff_sptr self*, *int priority*) → int

moving_average_ff_sptr.**thread_priority**(*moving_average_ff_sptr self*) → int

gnuradio.blocks.**moving_average_ii**(*int length*, *int scale*, *int max_iter=4096*, *unsigned int vlen=1*) → moving_average_ii_sptr

output is the moving sum of the last N samples, scaled by the scale factor

Constructor Specific Documentation:

Create a moving average block.

Parameters:
- **length** – Number of samples to use in the average.
- **scale** – scale factor for the result.
- **max_iter** – limits how long we go without flushing the accumulator This is necessary to avoid numerical instability for float and complex.
- **vlen** – When > 1, do a per-vector-element moving average

moving_average_ii_sptr.**active_thread_priority**(*moving_average_ii_sptr self*) → int

moving_average_ii_sptr.**declare_sample_delay**(*moving_average_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(moving_average_ii_sptr self, unsigned int delay)

moving_average_ii_sptr.**message_subscribers**(*moving_average_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

moving_average_ii_sptr.**min_noutput_items**(*moving_average_ii_sptr self*) → int

moving_average_ii_sptr.**pc_input_buffers_full_avg**(*moving_average_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(moving_average_ii_sptr self) -> pmt_vector_float

moving_average_ii_sptr.**pc_noutput_items_avg**(*moving_average_ii_sptr self*) → float

moving_average_ii_sptr.**pc_nproduced_avg**(*moving_average_ii_sptr self*) → float

moving_average_ii_sptr.**pc_output_buffers_full_avg**(*moving_average_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(moving_average_ii_sptr self) -> pmt_vector_float

moving_average_ii_sptr.**pc_throughput_avg**(*moving_average_ii_sptr self*) → float

moving_average_ii_sptr.**pc_work_time_avg**(*moving_average_ii_sptr self*) → float

moving_average_ii_sptr.**pc_work_time_total**(*moving_average_ii_sptr self*) → float

moving_average_ii_sptr.**sample_delay**(*moving_average_ii_sptr self*, *int which*) → unsigned int

moving_average_ii_sptr.**scale**(*moving_average_ii_sptr self*) → int

Get the scale factor being used.

moving_average_ii_sptr.**set_length**(*moving_average_ii_sptr self*, *int length*)

Set the length.

moving_average_ii_sptr.**set_length_and_scale**(*moving_average_ii_sptr self*, *int length*, *int scale*)

Set both the length and the scale factor together.

moving_average_ii_sptr.**set_min_noutput_items**(*moving_average_ii_sptr self*, *int m*)

moving_average_ii_sptr.**set_scale**(*moving_average_ii_sptr self*, *int scale*)

Set the scale factor.

moving_average_ii_sptr.**set_thread_priority**(*moving_average_ii_sptr self*, *int priority*) → int

moving_average_ii_sptr.**thread_priority**(*moving_average_ii_sptr self*) → int

gnuradio.blocks.**moving_average_ss**(*int length*, *short scale*, *int max_iter=4096*, *unsigned int vlen=1*) → moving_average_ss_sptr

output is the moving sum of the last N samples, scaled by the scale factor

Constructor Specific Documentation:

Create a moving average block.

Parameters:
- **length** – Number of samples to use in the average.
- **scale** – scale factor for the result.
- **max_iter** – limits how long we go without flushing the accumulator This is necessary to avoid numerical instability for float and complex.
- **vlen** – When > 1, do a per-vector-element moving average

moving_average_ss_sptr.**active_thread_priority**(*moving_average_ss_sptr self*) → int

moving_average_ss_sptr.**declare_sample_delay**(*moving_average_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(moving_average_ss_sptr self, unsigned int delay)

moving_average_ss_sptr.**message_subscribers**(*moving_average_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

moving_average_ss_sptr.**min_noutput_items**(*moving_average_ss_sptr self*) → int

moving_average_ss_sptr.**pc_input_buffers_full_avg**(*moving_average_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(moving_average_ss_sptr self) -> pmt_vector_float

moving_average_ss_sptr.**pc_noutput_items_avg**(*moving_average_ss_sptr self*) → float

moving_average_ss_sptr.**pc_nproduced_avg**(*moving_average_ss_sptr self*) → float

moving_average_ss_sptr.**pc_output_buffers_full_avg**(*moving_average_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(moving_average_ss_sptr self) -> pmt_vector_float

moving_average_ss_sptr.**pc_throughput_avg**(*moving_average_ss_sptr self*) → float

moving_average_ss_sptr.**pc_work_time_avg**(*moving_average_ss_sptr self*) → float

moving_average_ss_sptr.**pc_work_time_total**(*moving_average_ss_sptr self*) → float

moving_average_ss_sptr.**sample_delay**(*moving_average_ss_sptr self*, *int which*) → unsigned int

moving_average_ss_sptr.**scale**(*moving_average_ss_sptr self*) → short

Get the scale factor being used.

moving_average_ss_sptr.**set_length**(*moving_average_ss_sptr self*, *int length*)

Set the length.

moving_average_ss_sptr.**set_length_and_scale**(*moving_average_ss_sptr self*, *int length*, *short scale*)

Set both the length and the scale factor together.

moving_average_ss_sptr.**set_min_noutput_items**(*moving_average_ss_sptr self*, *int m*)

moving_average_ss_sptr.**set_scale**(*moving_average_ss_sptr self*, *short scale*)

Set the scale factor.

moving_average_ss_sptr.**set_thread_priority**(*moving_average_ss_sptr self*, *int priority*) → int

moving_average_ss_sptr.**thread_priority**(*moving_average_ss_sptr self*) → int

gnuradio.blocks.**multiply_cc**(*size_t vlen=1*) → multiply_cc_sptr

output = prod (input_0, input_1, ...)

Multiply across all input streams.

Constructor Specific Documentation:

Multiply streams of complex values.

Parameters: **vlen** – Vector length

multiply_cc_sptr.**active_thread_priority**(*multiply_cc_sptr self*) → int

multiply_cc_sptr.**declare_sample_delay**(*multiply_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_cc_sptr self, unsigned int delay)

multiply_cc_sptr.**message_subscribers**(*multiply_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_cc_sptr.**min_noutput_items**(*multiply_cc_sptr self*) → int

multiply_cc_sptr.**pc_input_buffers_full_avg**(*multiply_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_cc_sptr self) -> pmt_vector_float

multiply_cc_sptr.**pc_noutput_items_avg**(*multiply_cc_sptr self*) → float

multiply_cc_sptr.**pc_nproduced_avg**(*multiply_cc_sptr self*) → float

multiply_cc_sptr.**pc_output_buffers_full_avg**(*multiply_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_cc_sptr self) -> pmt_vector_float

multiply_cc_sptr.**pc_throughput_avg**(*multiply_cc_sptr self*) → float

multiply_cc_sptr.**pc_work_time_avg**(*multiply_cc_sptr self*) → float

multiply_cc_sptr.**pc_work_time_total**(*multiply_cc_sptr self*) → float

multiply_cc_sptr.**sample_delay**(*multiply_cc_sptr self*, *int which*) → unsigned int

multiply_cc_sptr.**set_min_noutput_items**(*multiply_cc_sptr self*, *int m*)

multiply_cc_sptr.**set_thread_priority**(*multiply_cc_sptr self*, *int priority*) → int

multiply_cc_sptr.**thread_priority**(*multiply_cc_sptr self*) → int

gnuradio.blocks.**multiply_conjugate_cc**(*size_t vlen=1*) → multiply_conjugate_cc_sptr

Multiplies stream 0 by the complex conjugate of stream 1.

Constructor Specific Documentation:

Multiplies a streams by the conjugate of a second stream.

Parameters: **vlen** – Vector length

multiply_conjugate_cc_sptr.**active_thread_priority**(*multiply_conjugate_cc_sptr self*) → int

multiply_conjugate_cc_sptr.**declare_sample_delay**(*multiply_conjugate_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_conjugate_cc_sptr self, unsigned int delay)

multiply_conjugate_cc_sptr.**message_subscribers**(*multiply_conjugate_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_conjugate_cc_sptr.**min_noutput_items**(*multiply_conjugate_cc_sptr self*) → int

multiply_conjugate_cc_sptr.**pc_input_buffers_full_avg**(*multiply_conjugate_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_conjugate_cc_sptr self) -> pmt_vector_float

multiply_conjugate_cc_sptr.**pc_noutput_items_avg**(*multiply_conjugate_cc_sptr self*) → float

multiply_conjugate_cc_sptr.**pc_nproduced_avg**(*multiply_conjugate_cc_sptr self*) → float

multiply_conjugate_cc_sptr.**pc_output_buffers_full_avg**(*multiply_conjugate_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_conjugate_cc_sptr self) -> pmt_vector_float

multiply_conjugate_cc_sptr.**pc_throughput_avg**(*multiply_conjugate_cc_sptr self*) → float

multiply_conjugate_cc_sptr.**pc_work_time_avg**(*multiply_conjugate_cc_sptr self*) → float

multiply_conjugate_cc_sptr.**pc_work_time_total**(*multiply_conjugate_cc_sptr self*) → float

multiply_conjugate_cc_sptr.**sample_delay**(*multiply_conjugate_cc_sptr self*, *int which*) → unsigned int

multiply_conjugate_cc_sptr.**set_min_noutput_items**(*multiply_conjugate_cc_sptr self*, *int m*)

multiply_conjugate_cc_sptr.**set_thread_priority**(*multiply_conjugate_cc_sptr self*, *int priority*) → int

multiply_conjugate_cc_sptr.**thread_priority**(*multiply_conjugate_cc_sptr self*) → int

gnuradio.blocks.**multiply_const_cc**(*gr_complex k*, *size_t vlen=1*) → multiply_const_cc_sptr

output = input * complex constant

Constructor Specific Documentation:

Create an instance of multiply_const_cc.

Parameters:
- **k** – complex multiplicative constant
- **vlen** – Vector length of incoming stream

multiply_const_cc_sptr.**active_thread_priority**(*multiply_const_cc_sptr self*) → int

multiply_const_cc_sptr.**declare_sample_delay**(*multiply_const_cc_sptr self*, *int which*, *int delay*)
declare_sample_delay(multiply_const_cc_sptr self, unsigned int delay)

multiply_const_cc_sptr.**k**(*multiply_const_cc_sptr self*) → gr_complex
Return complex multiplicative constant.

multiply_const_cc_sptr.**message_subscribers**(*multiply_const_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_cc_sptr.**min_noutput_items**(*multiply_const_cc_sptr self*) → int

multiply_const_cc_sptr.**pc_input_buffers_full_avg**(*multiply_const_cc_sptr self*, *int which*) → float
pc_input_buffers_full_avg(multiply_const_cc_sptr self) -> pmt_vector_float

multiply_const_cc_sptr.**pc_noutput_items_avg**(*multiply_const_cc_sptr self*) → float

multiply_const_cc_sptr.**pc_nproduced_avg**(*multiply_const_cc_sptr self*) → float

multiply_const_cc_sptr.**pc_output_buffers_full_avg**(*multiply_const_cc_sptr self*, *int which*) → float
pc_output_buffers_full_avg(multiply_const_cc_sptr self) -> pmt_vector_float

multiply_const_cc_sptr.**pc_throughput_avg**(*multiply_const_cc_sptr self*) → float

multiply_const_cc_sptr.**pc_work_time_avg**(*multiply_const_cc_sptr self*) → float

multiply_const_cc_sptr.**pc_work_time_total**(*multiply_const_cc_sptr self*) → float

multiply_const_cc_sptr.**sample_delay**(*multiply_const_cc_sptr self*, *int which*) → unsigned int

multiply_const_cc_sptr.**set_k**(*multiply_const_cc_sptr self*, *gr_complex k*)
Set complex multiplicative constant.

multiply_const_cc_sptr.**set_min_noutput_items**(*multiply_const_cc_sptr self*, *int m*)

multiply_const_cc_sptr.**set_thread_priority**(*multiply_const_cc_sptr self*, *int priority*) → int

multiply_const_cc_sptr.**thread_priority**(*multiply_const_cc_sptr self*) → int

gnuradio.blocks.**multiply_const_ff**(*float k*, *size_t vlen=1*) → multiply_const_ff_sptr
output = input * real constant

Constructor Specific Documentation:

Create an instance of multiply_const_ff.

Parameters:
- **k** – real multiplicative constant
- **vlen** – Vector length of incoming stream

multiply_const_ff_sptr.**active_thread_priority**(*multiply_const_ff_sptr self*) → int

multiply_const_ff_sptr.**declare_sample_delay**(*multiply_const_ff_sptr self*, *int which*, *int delay*)
declare_sample_delay(multiply_const_ff_sptr self, unsigned int delay)

multiply_const_ff_sptr.**k**(*multiply_const_ff_sptr self*) → float
Return real multiplicative constant.

multiply_const_ff_sptr.**message_subscribers**(*multiply_const_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_ff_sptr.**min_noutput_items**(*multiply_const_ff_sptr self*) → int

multiply_const_ff_sptr.**pc_input_buffers_full_avg**(*multiply_const_ff_sptr self*, *int which*) → float
pc_input_buffers_full_avg(multiply_const_ff_sptr self) -> pmt_vector_float

multiply_const_ff_sptr.**pc_noutput_items_avg**(*multiply_const_ff_sptr self*) → float

multiply_const_ff_sptr.**pc_nproduced_avg**(*multiply_const_ff_sptr self*) → float

multiply_const_ff_sptr.**pc_output_buffers_full_avg**(*multiply_const_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_const_ff_sptr self) -> pmt_vector_float

multiply_const_ff_sptr.**pc_throughput_avg**(*multiply_const_ff_sptr self*) → float

multiply_const_ff_sptr.**pc_work_time_avg**(*multiply_const_ff_sptr self*) → float

multiply_const_ff_sptr.**pc_work_time_total**(*multiply_const_ff_sptr self*) → float

multiply_const_ff_sptr.**sample_delay**(*multiply_const_ff_sptr self*, *int which*) → unsigned int

multiply_const_ff_sptr.**set_k**(*multiply_const_ff_sptr self*, *float k*)

Set real multiplicative constant.

multiply_const_ff_sptr.**set_min_noutput_items**(*multiply_const_ff_sptr self*, *int m*)

multiply_const_ff_sptr.**set_thread_priority**(*multiply_const_ff_sptr self*, *int priority*) → int

multiply_const_ff_sptr.**thread_priority**(*multiply_const_ff_sptr self*) → int

gnuradio.blocks.**multiply_const_ii**(*int k*) → multiply_const_ii_sptr

output = input * constant

Constructor Specific Documentation:

Create an instance of multiply_const_ii.

Parameters: **k** – multiplicative constant

multiply_const_ii_sptr.**active_thread_priority**(*multiply_const_ii_sptr self*) → int

multiply_const_ii_sptr.**declare_sample_delay**(*multiply_const_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_const_ii_sptr self, unsigned int delay)

multiply_const_ii_sptr.**k**(*multiply_const_ii_sptr self*) → int

Return multiplicative constant.

multiply_const_ii_sptr.**message_subscribers**(*multiply_const_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_ii_sptr.**min_noutput_items**(*multiply_const_ii_sptr self*) → int

multiply_const_ii_sptr.**pc_input_buffers_full_avg**(*multiply_const_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_const_ii_sptr self) -> pmt_vector_float

multiply_const_ii_sptr.**pc_noutput_items_avg**(*multiply_const_ii_sptr self*) → float

multiply_const_ii_sptr.**pc_nproduced_avg**(*multiply_const_ii_sptr self*) → float

multiply_const_ii_sptr.**pc_output_buffers_full_avg**(*multiply_const_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_const_ii_sptr self) -> pmt_vector_float

multiply_const_ii_sptr.**pc_throughput_avg**(*multiply_const_ii_sptr self*) → float

multiply_const_ii_sptr.**pc_work_time_avg**(*multiply_const_ii_sptr self*) → float

multiply_const_ii_sptr.**pc_work_time_total**(*multiply_const_ii_sptr self*) → float

multiply_const_ii_sptr.**sample_delay**(*multiply_const_ii_sptr self*, *int which*) → unsigned int

multiply_const_ii_sptr.**set_k**(*multiply_const_ii_sptr self*, *int k*)

Set multiplicative constant.

multiply_const_ii_sptr.**set_min_noutput_items**(*multiply_const_ii_sptr self*, *int m*)

multiply_const_ii_sptr.**set_thread_priority**(*multiply_const_ii_sptr self*, *int priority*) → int

multiply_const_ii_sptr.**thread_priority**(*multiply_const_ii_sptr self*) → int

gnuradio.blocks.**multiply_const_ss**(*short k*) → multiply_const_ss_sptr

output = input * constant

Constructor Specific Documentation:

Create an instance of multiply_const_ss.

Parameters: **k** – multiplicative constant

multiply_const_ss_sptr.**active_thread_priority**(*multiply_const_ss_sptr self*) → int

multiply_const_ss_sptr.**declare_sample_delay**(*multiply_const_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_const_ss_sptr self, unsigned int delay)

multiply_const_ss_sptr.**k**(*multiply_const_ss_sptr self*) → short

Return multiplicative constant.

multiply_const_ss_sptr.**message_subscribers**(*multiply_const_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_ss_sptr.**min_noutput_items**(*multiply_const_ss_sptr self*) → int

multiply_const_ss_sptr.**pc_input_buffers_full_avg**(*multiply_const_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_const_ss_sptr self) -> pmt_vector_float

multiply_const_ss_sptr.**pc_noutput_items_avg**(*multiply_const_ss_sptr self*) → float

multiply_const_ss_sptr.**pc_nproduced_avg**(*multiply_const_ss_sptr self*) → float

multiply_const_ss_sptr.**pc_output_buffers_full_avg**(*multiply_const_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_const_ss_sptr self) -> pmt_vector_float

multiply_const_ss_sptr.**pc_throughput_avg**(*multiply_const_ss_sptr self*) → float

multiply_const_ss_sptr.**pc_work_time_avg**(*multiply_const_ss_sptr self*) → float

multiply_const_ss_sptr.**pc_work_time_total**(*multiply_const_ss_sptr self*) → float

multiply_const_ss_sptr.**sample_delay**(*multiply_const_ss_sptr self*, *int which*) → unsigned int

multiply_const_ss_sptr.**set_k**(*multiply_const_ss_sptr self*, *short k*)

Set multiplicative constant.

multiply_const_ss_sptr.**set_min_noutput_items**(*multiply_const_ss_sptr self*, *int m*)

multiply_const_ss_sptr.**set_thread_priority**(*multiply_const_ss_sptr self*, *int priority*) → int

multiply_const_ss_sptr.**thread_priority**(*multiply_const_ss_sptr self*) → int

gnuradio.blocks.**multiply_const_vcc**(*pmt_vector_cfloat k*) → multiply_const_vcc_sptr

output = input * constant vector (element-wise)

Constructor Specific Documentation:

Create an instance of multiply_const_vcc.

Parameters: **k** – multiplicative constant vector

multiply_const_vcc_sptr.**active_thread_priority**(*multiply_const_vcc_sptr self*) → int

multiply_const_vcc_sptr.**declare_sample_delay**(*multiply_const_vcc_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_const_vcc_sptr self, unsigned int delay)

multiply_const_vcc_sptr.**k**(*multiply_const_vcc_sptr self*) → pmt_vector_cfloat

Return multiplicative constant vector.

multiply_const_vcc_sptr.**message_subscribers**(*multiply_const_vcc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_vcc_sptr.**min_noutput_items**(*multiply_const_vcc_sptr self*) → int

multiply_const_vcc_sptr.**pc_input_buffers_full_avg**(*multiply_const_vcc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_const_vcc_sptr self) -> pmt_vector_float

multiply_const_vcc_sptr.**pc_noutput_items_avg**(*multiply_const_vcc_sptr self*) → float

multiply_const_vcc_sptr.**pc_nproduced_avg**(*multiply_const_vcc_sptr self*) → float

multiply_const_vcc_sptr.**pc_output_buffers_full_avg**(*multiply_const_vcc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_const_vcc_sptr self) -> pmt_vector_float

multiply_const_vcc_sptr.**pc_throughput_avg**(*multiply_const_vcc_sptr self*) → float

multiply_const_vcc_sptr.**pc_work_time_avg**(*multiply_const_vcc_sptr self*) → float

multiply_const_vcc_sptr.**pc_work_time_total**(*multiply_const_vcc_sptr self*) → float

multiply_const_vcc_sptr.**sample_delay**(*multiply_const_vcc_sptr self*, *int which*) → unsigned int

multiply_const_vcc_sptr.**set_k**(*multiply_const_vcc_sptr self*, *pmt_vector_cfloat k*)

Set multiplicative constant vector.

multiply_const_vcc_sptr.**set_min_noutput_items**(*multiply_const_vcc_sptr self*, *int m*)

multiply_const_vcc_sptr.**set_thread_priority**(*multiply_const_vcc_sptr self*, *int priority*) → int

multiply_const_vcc_sptr.**thread_priority**(*multiply_const_vcc_sptr self*) → int

gnuradio.blocks.**multiply_const_vff**(*pmt_vector_float k*) → multiply_const_vff_sptr

output = input * constant vector (element-wise)

Constructor Specific Documentation:

Create an instance of multiply_const_vff.

Parameters: **k** – multiplicative constant vector

multiply_const_vff_sptr.**active_thread_priority**(*multiply_const_vff_sptr self*) → int

multiply_const_vff_sptr.**declare_sample_delay**(*multiply_const_vff_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_const_vff_sptr self, unsigned int delay)

multiply_const_vff_sptr.**k**(*multiply_const_vff_sptr self*) → pmt_vector_float

Return multiplicative constant vector.

multiply_const_vff_sptr.**message_subscribers**(*multiply_const_vff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_vff_sptr.**min_noutput_items**(*multiply_const_vff_sptr self*) → int

multiply_const_vff_sptr.**pc_input_buffers_full_avg**(*multiply_const_vff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_const_vff_sptr self) -> pmt_vector_float

multiply_const_vff_sptr.**pc_noutput_items_avg**(*multiply_const_vff_sptr self*) → float

multiply_const_vff_sptr.**pc_nproduced_avg**(*multiply_const_vff_sptr self*) → float

multiply_const_vff_sptr.**pc_output_buffers_full_avg**(*multiply_const_vff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_const_vff_sptr self) -> pmt_vector_float

multiply_const_vff_sptr.**pc_throughput_avg**(*multiply_const_vff_sptr self*) → float

multiply_const_vff_sptr.**pc_work_time_avg**(*multiply_const_vff_sptr self*) → float

multiply_const_vff_sptr.**pc_work_time_total**(*multiply_const_vff_sptr self*) → float

multiply_const_vff_sptr.**sample_delay**(*multiply_const_vff_sptr self*, *int which*) → unsigned int

multiply_const_vff_sptr.**set_k**(*multiply_const_vff_sptr self*, *pmt_vector_float k*)

Set multiplicative constant vector.

multiply_const_vff_sptr.**set_min_noutput_items**(*multiply_const_vff_sptr self*, *int m*)

multiply_const_vff_sptr.**set_thread_priority**(*multiply_const_vff_sptr self*, *int priority*) → int

multiply_const_vff_sptr.**thread_priority**(*multiply_const_vff_sptr self*) → int

gnuradio.blocks.**multiply_const_vii**(*std::vector< int, std::allocator< int > > k*) → multiply_const_vii_sptr

output = input * constant vector (element-wise)

Constructor Specific Documentation:

Create an instance of multiply_const_vii.

Parameters: **k** – multiplicative constant vector

multiply_const_vii_sptr.**active_thread_priority**(*multiply_const_vii_sptr self*) → int

multiply_const_vii_sptr.**declare_sample_delay**(*multiply_const_vii_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_const_vii_sptr self, unsigned int delay)

multiply_const_vii_sptr.**k**(*multiply_const_vii_sptr self*) → std::vector< int,std::allocator< int > >

Return multiplicative constant vector.

multiply_const_vii_sptr.**message_subscribers**(*multiply_const_vii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_vii_sptr.**min_noutput_items**(*multiply_const_vii_sptr self*) → int

multiply_const_vii_sptr.**pc_input_buffers_full_avg**(*multiply_const_vii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_const_vii_sptr self) -> pmt_vector_float

multiply_const_vii_sptr.**pc_noutput_items_avg**(*multiply_const_vii_sptr self*) → float

multiply_const_vii_sptr.**pc_nproduced_avg**(*multiply_const_vii_sptr self*) → float

multiply_const_vii_sptr.**pc_output_buffers_full_avg**(*multiply_const_vii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_const_vii_sptr self) -> pmt_vector_float

multiply_const_vii_sptr.**pc_throughput_avg**(*multiply_const_vii_sptr self*) → float

multiply_const_vii_sptr.**pc_work_time_avg**(*multiply_const_vii_sptr self*) → float

multiply_const_vii_sptr.**pc_work_time_total**(*multiply_const_vii_sptr self*) → float

multiply_const_vii_sptr.**sample_delay**(*multiply_const_vii_sptr self*, *int which*) → unsigned int

multiply_const_vii_sptr.**set_k**(*multiply_const_vii_sptr self*, *std::vector< int, std::allocator< int > > k*)

Set multiplicative constant vector.

multiply_const_vii_sptr.**set_min_noutput_items**(*multiply_const_vii_sptr self*, *int m*)

multiply_const_vii_sptr.**set_thread_priority**(*multiply_const_vii_sptr self*, *int priority*) → int

multiply_const_vii_sptr.**thread_priority**(*multiply_const_vii_sptr self*) → int

gnuradio.blocks.**multiply_const_vss**(*std::vector< short, std::allocator< short > > k*) → multiply_const_vss_sptr

output = input * constant vector (element-wise)

Constructor Specific Documentation:

Create an instance of multiply_const_vss.

Parameters: **k** – multiplicative constant vector

multiply_const_vss_sptr.**active_thread_priority**(*multiply_const_vss_sptr self*) → int

multiply_const_vss_sptr.**declare_sample_delay**(*multiply_const_vss_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_const_vss_sptr self, unsigned int delay)

multiply_const_vss_sptr.**k**(*multiply_const_vss_sptr self*) → std::vector< short,std::allocator< short > >

Return multiplicative constant vector.

multiply_const_vss_sptr.**message_subscribers**(*multiply_const_vss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_const_vss_sptr.**min_noutput_items**(*multiply_const_vss_sptr self*) → int

multiply_const_vss_sptr.**pc_input_buffers_full_avg**(*multiply_const_vss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_const_vss_sptr self) -> pmt_vector_float

multiply_const_vss_sptr.**pc_noutput_items_avg**(*multiply_const_vss_sptr self*) → float

multiply_const_vss_sptr.**pc_nproduced_avg**(*multiply_const_vss_sptr self*) → float

multiply_const_vss_sptr.**pc_output_buffers_full_avg**(*multiply_const_vss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_const_vss_sptr self) -> pmt_vector_float

multiply_const_vss_sptr.**pc_throughput_avg**(*multiply_const_vss_sptr self*) → float

multiply_const_vss_sptr.**pc_work_time_avg**(*multiply_const_vss_sptr self*) → float

multiply_const_vss_sptr.**pc_work_time_total**(*multiply_const_vss_sptr self*) → float

multiply_const_vss_sptr.**sample_delay**(*multiply_const_vss_sptr self*, *int which*) → unsigned int

multiply_const_vss_sptr.**set_k**(*multiply_const_vss_sptr self*, *std::vector< short, std::allocator< short > > k*)

Set multiplicative constant vector.

multiply_const_vss_sptr.**set_min_noutput_items**(*multiply_const_vss_sptr self*, *int m*)

multiply_const_vss_sptr.**set_thread_priority**(*multiply_const_vss_sptr self*, *int priority*) → int

multiply_const_vss_sptr.**thread_priority**(*multiply_const_vss_sptr self*) → int

gnuradio.blocks.**multiply_ff**(*size_t vlen=1*) → multiply_ff_sptr

output = prod (input_0, input_1, ...)

Multiply across all input streams.

Constructor Specific Documentation:

Multiply streams of float values.

Parameters: **vlen** – Vector length

multiply_ff_sptr.**active_thread_priority**(*multiply_ff_sptr self*) → int

multiply_ff_sptr.**declare_sample_delay**(*multiply_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_ff_sptr self, unsigned int delay)

multiply_ff_sptr.**message_subscribers**(*multiply_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_ff_sptr.**min_noutput_items**(*multiply_ff_sptr self*) → int

multiply_ff_sptr.**pc_input_buffers_full_avg**(*multiply_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_ff_sptr self) -> pmt_vector_float

multiply_ff_sptr.**pc_noutput_items_avg**(*multiply_ff_sptr self*) → float

multiply_ff_sptr.**pc_nproduced_avg**(*multiply_ff_sptr self*) → float

multiply_ff_sptr.**pc_output_buffers_full_avg**(*multiply_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_ff_sptr self) -> pmt_vector_float

multiply_ff_sptr.**pc_throughput_avg**(*multiply_ff_sptr self*) → float

multiply_ff_sptr.**pc_work_time_avg**(*multiply_ff_sptr self*) → float

multiply_ff_sptr.**pc_work_time_total**(*multiply_ff_sptr self*) → float

multiply_ff_sptr.**sample_delay**(*multiply_ff_sptr self*, *int which*) → unsigned int

multiply_ff_sptr.**set_min_noutput_items**(*multiply_ff_sptr self*, *int m*)

multiply_ff_sptr.**set_thread_priority**(*multiply_ff_sptr self*, *int priority*) → int

multiply_ff_sptr.**thread_priority**(*multiply_ff_sptr self*) → int

gnuradio.blocks.**multiply_ii**(*size_t vlen=1*) → multiply_ii_sptr

output = prod (input_0, input_1, ...)

Multiply across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

multiply_ii_sptr.**active_thread_priority**(*multiply_ii_sptr self*) → int

multiply_ii_sptr.**declare_sample_delay**(*multiply_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_ii_sptr self, unsigned int delay)

multiply_ii_sptr.**message_subscribers**(*multiply_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_ii_sptr.**min_noutput_items**(*multiply_ii_sptr self*) → int

multiply_ii_sptr.**pc_input_buffers_full_avg**(*multiply_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_ii_sptr self) -> pmt_vector_float

multiply_ii_sptr.**pc_noutput_items_avg**(*multiply_ii_sptr self*) → float

multiply_ii_sptr.**pc_nproduced_avg**(*multiply_ii_sptr self*) → float

multiply_ii_sptr.**pc_output_buffers_full_avg**(*multiply_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_ii_sptr self) -> pmt_vector_float

multiply_ii_sptr.**pc_throughput_avg**(*multiply_ii_sptr self*) → float

multiply_ii_sptr.**pc_work_time_avg**(*multiply_ii_sptr self*) → float

multiply_ii_sptr.**pc_work_time_total**(*multiply_ii_sptr self*) → float

multiply_ii_sptr.**sample_delay**(*multiply_ii_sptr self*, *int which*) → unsigned int

multiply_ii_sptr.**set_min_noutput_items**(*multiply_ii_sptr self*, *int m*)

multiply_ii_sptr.**set_thread_priority**(*multiply_ii_sptr self*, *int priority*) → int

multiply_ii_sptr.**thread_priority**(*multiply_ii_sptr self*) → int

gnuradio.blocks.**multiply_matrix_ff**(*std::vector< std::vector< float, std::allocator< float > >, std::allocator< std::vector< float, std::allocator< float > > > > A*, *gr::block::tag_propagation_policy_t tag_propagation_policy*) → multiply_matrix_ff_sptr

Matrix multiplexer/multiplier: y(k) = A x(k)

This block is similar to gr::blocks::multiply_const_ff, the difference being it can handle several inputs and outputs, and the input-to-output relation can be described by the following mathematical equation: and are column-vectors describing the elements on the input port at time step (this is a sync block with no memory).

Examples for where to use this block include:

This block features a special tag propagation mode: When setting the tag propagation policy to gr::block::TPP_CUSTOM, a tag is propagated from input to output , if .

Message Ports This block as one input message port (). A message sent to this port will be converted to a std::vector<std::vector<float> >, and then passed on to set_A(). If no conversion is possible, a warning is issued via the logging interface, and A remains unchanged.

: It is not possible to change the dimension of the matrix after initialization, as this affects the I/O signature! If a matrix of invalid size is passed to the block, an alert is raised via the logging interface, and A remains unchanged.

Constructor Specific Documentation:

Parameters:
- **A** – The matrix
- **tag_propagation_policy** – The tag propagation policy. Note this can be any gr::block::tag_propagation_policy_t value. In case of TPP_CUSTOM, tags are only transferred from input to output .

multiply_matrix_ff_sptr.**MSG_PORT_NAME_SET_A**(*multiply_matrix_ff_sptr self*) → std::string const &

multiply_matrix_ff_sptr.**active_thread_priority**(*multiply_matrix_ff_sptr self*) → int

multiply_matrix_ff_sptr.**declare_sample_delay**(*multiply_matrix_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_matrix_ff_sptr self, unsigned int delay)

multiply_matrix_ff_sptr.**get_A**(*multiply_matrix_ff_sptr self*) → std::vector< std::vector< float,std::allocator< float > >,std::allocator< std::vector< float,std::allocator< float > > > > const &

Returns the current matrix.

multiply_matrix_ff_sptr.**message_subscribers**(*multiply_matrix_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_matrix_ff_sptr.**min_noutput_items**(*multiply_matrix_ff_sptr self*) → int

multiply_matrix_ff_sptr.**pc_input_buffers_full_avg**(*multiply_matrix_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_matrix_ff_sptr self) -> pmt_vector_float

multiply_matrix_ff_sptr.**pc_noutput_items_avg**(*multiply_matrix_ff_sptr self*) → float

multiply_matrix_ff_sptr.**pc_nproduced_avg**(*multiply_matrix_ff_sptr self*) → float

multiply_matrix_ff_sptr.**pc_output_buffers_full_avg**(*multiply_matrix_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_matrix_ff_sptr self) -> pmt_vector_float

multiply_matrix_ff_sptr.**pc_throughput_avg**(*multiply_matrix_ff_sptr self*) → float

multiply_matrix_ff_sptr.**pc_work_time_avg**(*multiply_matrix_ff_sptr self*) → float

multiply_matrix_ff_sptr.**pc_work_time_total**(*multiply_matrix_ff_sptr self*) → float

multiply_matrix_ff_sptr.**sample_delay**(*multiply_matrix_ff_sptr self*, *int which*) → unsigned int

multiply_matrix_ff_sptr.**set_A**(*multiply_matrix_ff_sptr self*, *std::vector< std::vector< float, std::allocator< float > >, std::allocator< std::vector< float, std::allocator< float > > > > const & new_A*) → bool

Sets the matrix to a new value . Returns true if the new matrix was valid and could be changed.

multiply_matrix_ff_sptr.**set_min_noutput_items**(*multiply_matrix_ff_sptr self*, *int m*)

multiply_matrix_ff_sptr.**set_thread_priority**(*multiply_matrix_ff_sptr self*, *int priority*) → int

multiply_matrix_ff_sptr.**thread_priority**(*multiply_matrix_ff_sptr self*) → int

gnuradio.blocks.**multiply_ss**(*size_t vlen=1*) → multiply_ss_sptr

output = prod (input_0, input_1, ...)

Multiply across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

multiply_ss_sptr.**active_thread_priority**(*multiply_ss_sptr self*) → int

multiply_ss_sptr.**declare_sample_delay**(*multiply_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(multiply_ss_sptr self, unsigned int delay)

multiply_ss_sptr.**message_subscribers**(*multiply_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

multiply_ss_sptr.**min_noutput_items**(*multiply_ss_sptr self*) → int

multiply_ss_sptr.**pc_input_buffers_full_avg**(*multiply_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(multiply_ss_sptr self) -> pmt_vector_float

multiply_ss_sptr.**pc_noutput_items_avg**(*multiply_ss_sptr self*) → float

multiply_ss_sptr.**pc_nproduced_avg**(*multiply_ss_sptr self*) → float

multiply_ss_sptr.**pc_output_buffers_full_avg**(*multiply_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(multiply_ss_sptr self) -> pmt_vector_float

multiply_ss_sptr.**pc_throughput_avg**(*multiply_ss_sptr self*) → float

multiply_ss_sptr.**pc_work_time_avg**(*multiply_ss_sptr self*) → float

multiply_ss_sptr.**pc_work_time_total**(*multiply_ss_sptr self*) → float

multiply_ss_sptr.**sample_delay**(*multiply_ss_sptr self*, *int which*) → unsigned int

multiply_ss_sptr.**set_min_noutput_items**(*multiply_ss_sptr self*, *int m*)

multiply_ss_sptr.**set_thread_priority**(*multiply_ss_sptr self*, *int priority*) → int

multiply_ss_sptr.**thread_priority**(*multiply_ss_sptr self*) → int

gnuradio.blocks.**mute_cc**(*bool mute=False*) → mute_cc_sptr

output = input or zero if muted.

Constructor Specific Documentation:

Parameters: **mute** –

mute_cc_sptr.**active_thread_priority**(*mute_cc_sptr self*) → int

mute_cc_sptr.**declare_sample_delay**(*mute_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(mute_cc_sptr self, unsigned int delay)

mute_cc_sptr.**message_subscribers**(*mute_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

mute_cc_sptr.**min_noutput_items**(*mute_cc_sptr self*) → int

mute_cc_sptr.**mute**(*mute_cc_sptr self*) → bool

mute_cc_sptr.**pc_input_buffers_full_avg**(*mute_cc_sptr self*, *int which*) → float
pc_input_buffers_full_avg(mute_cc_sptr self) -> pmt_vector_float

mute_cc_sptr.**pc_noutput_items_avg**(*mute_cc_sptr self*) → float

mute_cc_sptr.**pc_nproduced_avg**(*mute_cc_sptr self*) → float

mute_cc_sptr.**pc_output_buffers_full_avg**(*mute_cc_sptr self*, *int which*) → float
pc_output_buffers_full_avg(mute_cc_sptr self) -> pmt_vector_float

mute_cc_sptr.**pc_throughput_avg**(*mute_cc_sptr self*) → float

mute_cc_sptr.**pc_work_time_avg**(*mute_cc_sptr self*) → float

mute_cc_sptr.**pc_work_time_total**(*mute_cc_sptr self*) → float

mute_cc_sptr.**sample_delay**(*mute_cc_sptr self*, *int which*) → unsigned int

mute_cc_sptr.**set_min_noutput_items**(*mute_cc_sptr self*, *int m*)

mute_cc_sptr.**set_mute**(*mute_cc_sptr self*, *bool mute=False*)

mute_cc_sptr.**set_thread_priority**(*mute_cc_sptr self*, *int priority*) → int

mute_cc_sptr.**thread_priority**(*mute_cc_sptr self*) → int

gnuradio.blocks.**mute_ff**(*bool mute=False*) → mute_ff_sptr
output = input or zero if muted.

Constructor Specific Documentation:

Parameters: **mute** –

mute_ff_sptr.**active_thread_priority**(*mute_ff_sptr self*) → int

mute_ff_sptr.**declare_sample_delay**(*mute_ff_sptr self*, *int which*, *int delay*)
declare_sample_delay(mute_ff_sptr self, unsigned int delay)

mute_ff_sptr.**message_subscribers**(*mute_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

mute_ff_sptr.**min_noutput_items**(*mute_ff_sptr self*) → int

mute_ff_sptr.**mute**(*mute_ff_sptr self*) → bool

mute_ff_sptr.**pc_input_buffers_full_avg**(*mute_ff_sptr self*, *int which*) → float
pc_input_buffers_full_avg(mute_ff_sptr self) -> pmt_vector_float

mute_ff_sptr.**pc_noutput_items_avg**(*mute_ff_sptr self*) → float

mute_ff_sptr.**pc_nproduced_avg**(*mute_ff_sptr self*) → float

mute_ff_sptr.**pc_output_buffers_full_avg**(*mute_ff_sptr self*, *int which*) → float
pc_output_buffers_full_avg(mute_ff_sptr self) -> pmt_vector_float

mute_ff_sptr.**pc_throughput_avg**(*mute_ff_sptr self*) → float

mute_ff_sptr.**pc_work_time_avg**(*mute_ff_sptr self*) → float

mute_ff_sptr.**pc_work_time_total**(*mute_ff_sptr self*) → float

mute_ff_sptr.**sample_delay**(*mute_ff_sptr self*, *int which*) → unsigned int

mute_ff_sptr.**set_min_noutput_items**(*mute_ff_sptr self*, *int m*)

mute_ff_sptr.**set_mute**(*mute_ff_sptr self*, *bool mute=False*)

mute_ff_sptr.**set_thread_priority**(*mute_ff_sptr self*, *int priority*) → int

mute_ff_sptr.**thread_priority**(*mute_ff_sptr self*) → int

gnuradio.blocks.**mute_ii**(*bool mute=False*) → mute_ii_sptr

output = input or zero if muted.

Constructor Specific Documentation:

Parameters: **mute** –

mute_ii_sptr.**active_thread_priority**(*mute_ii_sptr self*) → int

mute_ii_sptr.**declare_sample_delay**(*mute_ii_sptr self*, *int which*, *int delay*)
declare_sample_delay(mute_ii_sptr self, unsigned int delay)

mute_ii_sptr.**message_subscribers**(*mute_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

mute_ii_sptr.**min_noutput_items**(*mute_ii_sptr self*) → int

mute_ii_sptr.**mute**(*mute_ii_sptr self*) → bool

mute_ii_sptr.**pc_input_buffers_full_avg**(*mute_ii_sptr self*, *int which*) → float
pc_input_buffers_full_avg(mute_ii_sptr self) -> pmt_vector_float

mute_ii_sptr.**pc_noutput_items_avg**(*mute_ii_sptr self*) → float

mute_ii_sptr.**pc_nproduced_avg**(*mute_ii_sptr self*) → float

mute_ii_sptr.**pc_output_buffers_full_avg**(*mute_ii_sptr self*, *int which*) → float
pc_output_buffers_full_avg(mute_ii_sptr self) -> pmt_vector_float

mute_ii_sptr.**pc_throughput_avg**(*mute_ii_sptr self*) → float

mute_ii_sptr.**pc_work_time_avg**(*mute_ii_sptr self*) → float

mute_ii_sptr.**pc_work_time_total**(*mute_ii_sptr self*) → float

mute_ii_sptr.**sample_delay**(*mute_ii_sptr self*, *int which*) → unsigned int

mute_ii_sptr.**set_min_noutput_items**(*mute_ii_sptr self*, *int m*)

mute_ii_sptr.**set_mute**(*mute_ii_sptr self*, *bool mute=False*)

mute_ii_sptr.**set_thread_priority**(*mute_ii_sptr self*, *int priority*) → int

mute_ii_sptr.**thread_priority**(*mute_ii_sptr self*) → int

gnuradio.blocks.**mute_ss**(*bool mute=False*) → mute_ss_sptr
output = input or zero if muted.

Constructor Specific Documentation:

Parameters: **mute** –

mute_ss_sptr.**active_thread_priority**(*mute_ss_sptr self*) → int

mute_ss_sptr.**declare_sample_delay**(*mute_ss_sptr self*, *int which*, *int delay*)
declare_sample_delay(mute_ss_sptr self, unsigned int delay)

mute_ss_sptr.**message_subscribers**(*mute_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

mute_ss_sptr.**min_noutput_items**(*mute_ss_sptr self*) → int

mute_ss_sptr.**mute**(*mute_ss_sptr self*) → bool

mute_ss_sptr.**pc_input_buffers_full_avg**(*mute_ss_sptr self*, *int which*) → float
pc_input_buffers_full_avg(mute_ss_sptr self) -> pmt_vector_float

mute_ss_sptr.**pc_noutput_items_avg**(*mute_ss_sptr self*) → float

mute_ss_sptr.**pc_nproduced_avg**(*mute_ss_sptr self*) → float

mute_ss_sptr.**pc_output_buffers_full_avg**(*mute_ss_sptr self*, *int which*) → float
pc_output_buffers_full_avg(mute_ss_sptr self) -> pmt_vector_float

mute_ss_sptr.**pc_throughput_avg**(*mute_ss_sptr self*) → float

mute_ss_sptr.**pc_work_time_avg**(*mute_ss_sptr self*) → float

mute_ss_sptr.**pc_work_time_total**(*mute_ss_sptr self*) → float

mute_ss_sptr.**sample_delay**(*mute_ss_sptr self*, *int which*) → unsigned int

mute_ss_sptr.**set_min_noutput_items**(*mute_ss_sptr self*, *int m*)

mute_ss_sptr.**set_mute**(*mute_ss_sptr self*, *bool mute=False*)

mute_ss_sptr.**set_thread_priority**(*mute_ss_sptr self*, *int priority*) → int

mute_ss_sptr.**thread_priority**(*mute_ss_sptr self*) → int

gnuradio.blocks.**nlog10_ff**(*float n=1.0*, *size_t vlen=1*, *float k=0.0*) → nlog10_ff_sptr

output = n*log10(input) + k

Constructor Specific Documentation:

Make an instance of an nlog10_ff block.

Parameters:
- **n** – Scalar multiplicative constant
- **vlen** – Input vector length
- **k** – Scalar additive constant

nlog10_ff_sptr.**active_thread_priority**(*nlog10_ff_sptr self*) → int

nlog10_ff_sptr.**declare_sample_delay**(*nlog10_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(nlog10_ff_sptr self, unsigned int delay)

nlog10_ff_sptr.**message_subscribers**(*nlog10_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

nlog10_ff_sptr.**min_noutput_items**(*nlog10_ff_sptr self*) → int

nlog10_ff_sptr.**pc_input_buffers_full_avg**(*nlog10_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(nlog10_ff_sptr self) -> pmt_vector_float

nlog10_ff_sptr.**pc_noutput_items_avg**(*nlog10_ff_sptr self*) → float

nlog10_ff_sptr.**pc_nproduced_avg**(*nlog10_ff_sptr self*) → float

nlog10_ff_sptr.**pc_output_buffers_full_avg**(*nlog10_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(nlog10_ff_sptr self) -> pmt_vector_float

nlog10_ff_sptr.**pc_throughput_avg**(*nlog10_ff_sptr self*) → float

nlog10_ff_sptr.**pc_work_time_avg**(*nlog10_ff_sptr self*) → float

nlog10_ff_sptr.**pc_work_time_total**(*nlog10_ff_sptr self*) → float

nlog10_ff_sptr.**sample_delay**(*nlog10_ff_sptr self*, *int which*) → unsigned int

nlog10_ff_sptr.**set_min_noutput_items**(*nlog10_ff_sptr self*, *int m*)

nlog10_ff_sptr.**set_thread_priority**(*nlog10_ff_sptr self*, *int priority*) → int

nlog10_ff_sptr.**thread_priority**(*nlog10_ff_sptr self*) → int

gnuradio.blocks.**nop**(*size_t sizeof_stream_item*) → nop_sptr

Does nothing. Used for testing only.

Constructor Specific Documentation:

Build a nop block.

Parameters: **sizeof_stream_item** – size of the stream items in bytes.

nop_sptr.**active_thread_priority**(*nop_sptr self*) → int

nop_sptr.**ctrlport_test**(*nop_sptr self*) → int

nop_sptr.**declare_sample_delay**(*nop_sptr self*, *int which*, *int delay*)

declare_sample_delay(nop_sptr self, unsigned int delay)

nop_sptr.**message_subscribers**(*nop_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

nop_sptr.**min_noutput_items**(*nop_sptr self*) → int

nop_sptr.**nmsgs_received**(*nop_sptr self*) → int

nop_sptr.**pc_input_buffers_full_avg**(*nop_sptr self*, *int which*) → float

pc_input_buffers_full_avg(nop_sptr self) -> pmt_vector_float

nop_sptr.**pc_noutput_items_avg**(*nop_sptr self*) → float

nop_sptr.**pc_nproduced_avg**(*nop_sptr self*) → float

nop_sptr.**pc_output_buffers_full_avg**(*nop_sptr self*, *int which*) → float
pc_output_buffers_full_avg(nop_sptr self) -> pmt_vector_float

nop_sptr.**pc_throughput_avg**(*nop_sptr self*) → float

nop_sptr.**pc_work_time_avg**(*nop_sptr self*) → float

nop_sptr.**pc_work_time_total**(*nop_sptr self*) → float

nop_sptr.**sample_delay**(*nop_sptr self*, *int which*) → unsigned int

nop_sptr.**set_ctrlport_test**(*nop_sptr self*, *int x*)

nop_sptr.**set_min_noutput_items**(*nop_sptr self*, *int m*)

nop_sptr.**set_thread_priority**(*nop_sptr self*, *int priority*) → int

nop_sptr.**thread_priority**(*nop_sptr self*) → int

gnuradio.blocks.**not_bb**(*size_t vlen=1*) → not_bb_sptr
output = ~input

bitwise boolean not of input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

not_bb_sptr.**active_thread_priority**(*not_bb_sptr self*) → int

not_bb_sptr.**declare_sample_delay**(*not_bb_sptr self*, *int which*, *int delay*)
declare_sample_delay(not_bb_sptr self, unsigned int delay)

not_bb_sptr.**message_subscribers**(*not_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

not_bb_sptr.**min_noutput_items**(*not_bb_sptr self*) → int

not_bb_sptr.**pc_input_buffers_full_avg**(*not_bb_sptr self*, *int which*) → float
pc_input_buffers_full_avg(not_bb_sptr self) -> pmt_vector_float

not_bb_sptr.**pc_noutput_items_avg**(*not_bb_sptr self*) → float

not_bb_sptr.**pc_nproduced_avg**(*not_bb_sptr self*) → float

not_bb_sptr.**pc_output_buffers_full_avg**(*not_bb_sptr self*, *int which*) → float
pc_output_buffers_full_avg(not_bb_sptr self) -> pmt_vector_float

not_bb_sptr.**pc_throughput_avg**(*not_bb_sptr self*) → float

not_bb_sptr.**pc_work_time_avg**(*not_bb_sptr self*) → float

not_bb_sptr.**pc_work_time_total**(*not_bb_sptr self*) → float

not_bb_sptr.**sample_delay**(*not_bb_sptr self*, *int which*) → unsigned int

not_bb_sptr.**set_min_noutput_items**(*not_bb_sptr self*, *int m*)

not_bb_sptr.**set_thread_priority**(*not_bb_sptr self*, *int priority*) → int

not_bb_sptr.**thread_priority**(*not_bb_sptr self*) → int

gnuradio.blocks.**not_ii**(*size_t vlen=1*) → not_ii_sptr
output = ~input

bitwise boolean not of input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

not_ii_sptr.**active_thread_priority**(*not_ii_sptr self*) → int

not_ii_sptr.**declare_sample_delay**(*not_ii_sptr self*, *int which*, *int delay*)
declare_sample_delay(not_ii_sptr self, unsigned int delay)

not_ii_sptr.**message_subscribers**(*not_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

not_ii_sptr.**min_noutput_items**(*not_ii_sptr self*) → int

not_ii_sptr.**pc_input_buffers_full_avg**(*not_ii_sptr self*, *int which*) → float
pc_input_buffers_full_avg(not_ii_sptr self) -> pmt_vector_float

not_ii_sptr.**pc_noutput_items_avg**(*not_ii_sptr self*) → float

not_ii_sptr.**pc_nproduced_avg**(*not_ii_sptr self*) → float

not_ii_sptr.**pc_output_buffers_full_avg**(*not_ii_sptr self*, *int which*) → float
pc_output_buffers_full_avg(not_ii_sptr self) -> pmt_vector_float

not_ii_sptr.**pc_throughput_avg**(*not_ii_sptr self*) → float

not_ii_sptr.**pc_work_time_avg**(*not_ii_sptr self*) → float

not_ii_sptr.**pc_work_time_total**(*not_ii_sptr self*) → float

not_ii_sptr.**sample_delay**(*not_ii_sptr self*, *int which*) → unsigned int

not_ii_sptr.**set_min_noutput_items**(*not_ii_sptr self*, *int m*)

not_ii_sptr.**set_thread_priority**(*not_ii_sptr self*, *int priority*) → int

not_ii_sptr.**thread_priority**(*not_ii_sptr self*) → int

gnuradio.blocks.**not_ss**(*size_t vlen=1*) → not_ss_sptr
output = ~input

bitwise boolean not of input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

not_ss_sptr.**active_thread_priority**(*not_ss_sptr self*) → int

not_ss_sptr.**declare_sample_delay**(*not_ss_sptr self*, *int which*, *int delay*)
declare_sample_delay(not_ss_sptr self, unsigned int delay)

not_ss_sptr.**message_subscribers**(*not_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

not_ss_sptr.**min_noutput_items**(*not_ss_sptr self*) → int

not_ss_sptr.**pc_input_buffers_full_avg**(*not_ss_sptr self*, *int which*) → float
pc_input_buffers_full_avg(not_ss_sptr self) -> pmt_vector_float

not_ss_sptr.**pc_noutput_items_avg**(*not_ss_sptr self*) → float

not_ss_sptr.**pc_nproduced_avg**(*not_ss_sptr self*) → float

not_ss_sptr.**pc_output_buffers_full_avg**(*not_ss_sptr self*, *int which*) → float
pc_output_buffers_full_avg(not_ss_sptr self) -> pmt_vector_float

not_ss_sptr.**pc_throughput_avg**(*not_ss_sptr self*) → float

not_ss_sptr.**pc_work_time_avg**(*not_ss_sptr self*) → float

not_ss_sptr.**pc_work_time_total**(*not_ss_sptr self*) → float

not_ss_sptr.**sample_delay**(*not_ss_sptr self*, *int which*) → unsigned int

not_ss_sptr.**set_min_noutput_items**(*not_ss_sptr self*, *int m*)

not_ss_sptr.**set_thread_priority**(*not_ss_sptr self*, *int priority*) → int

not_ss_sptr.**thread_priority**(*not_ss_sptr self*) → int

gnuradio.blocks.**null_sink**(*size_t sizeof_stream_item*) → null_sink_sptr
Bit bucket. Use as a termination point when a sink is required and we don't want to do anything real.

Constructor Specific Documentation:

Build a null sink block.

Parameters: **sizeof_stream_item** – size of the stream items in bytes.

null_sink_sptr.**active_thread_priority**(*null_sink_sptr self*) → int

null_sink_sptr.**declare_sample_delay**(*null_sink_sptr self*, *int which*, *int delay*)
declare_sample_delay(null_sink_sptr self, unsigned int delay)

null_sink_sptr.**message_subscribers**(*null_sink_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

null_sink_sptr.**min_noutput_items**(*null_sink_sptr self*) → int

null_sink_sptr.**pc_input_buffers_full_avg**(*null_sink_sptr self*, *int which*) → float
pc_input_buffers_full_avg(null_sink_sptr self) -> pmt_vector_float

null_sink_sptr.**pc_noutput_items_avg**(*null_sink_sptr self*) → float

null_sink_sptr.**pc_nproduced_avg**(*null_sink_sptr self*) → float

null_sink_sptr.**pc_output_buffers_full_avg**(*null_sink_sptr self*, *int which*) → float
pc_output_buffers_full_avg(null_sink_sptr self) -> pmt_vector_float

null_sink_sptr.**pc_throughput_avg**(*null_sink_sptr self*) → float

null_sink_sptr.**pc_work_time_avg**(*null_sink_sptr self*) → float

null_sink_sptr.**pc_work_time_total**(*null_sink_sptr self*) → float

null_sink_sptr.**sample_delay**(*null_sink_sptr self*, *int which*) → unsigned int

null_sink_sptr.**set_min_noutput_items**(*null_sink_sptr self*, *int m*)

null_sink_sptr.**set_thread_priority**(*null_sink_sptr self*, *int priority*) → int

null_sink_sptr.**thread_priority**(*null_sink_sptr self*) → int

gnuradio.blocks.**null_source**(*size_t sizeof_stream_item*) → null_source_sptr
A source of zeros used mainly for testing.

Constructor Specific Documentation:

Build a null source block.

Parameters: **sizeof_stream_item** – size of the stream items in bytes.

null_source_sptr.**active_thread_priority**(*null_source_sptr self*) → int

null_source_sptr.**declare_sample_delay**(*null_source_sptr self*, *int which*, *int delay*)
declare_sample_delay(null_source_sptr self, unsigned int delay)

null_source_sptr.**message_subscribers**(*null_source_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

null_source_sptr.**min_noutput_items**(*null_source_sptr self*) → int

null_source_sptr.**pc_input_buffers_full_avg**(*null_source_sptr self*, *int which*) → float
pc_input_buffers_full_avg(null_source_sptr self) -> pmt_vector_float

null_source_sptr.**pc_noutput_items_avg**(*null_source_sptr self*) → float

null_source_sptr.**pc_nproduced_avg**(*null_source_sptr self*) → float

null_source_sptr.**pc_output_buffers_full_avg**(*null_source_sptr self*, *int which*) → float
pc_output_buffers_full_avg(null_source_sptr self) -> pmt_vector_float

null_source_sptr.**pc_throughput_avg**(*null_source_sptr self*) → float

null_source_sptr.**pc_work_time_avg**(*null_source_sptr self*) → float

null_source_sptr.**pc_work_time_total**(*null_source_sptr self*) → float

null_source_sptr.**sample_delay**(*null_source_sptr self*, *int which*) → unsigned int

null_source_sptr.**set_min_noutput_items**(*null_source_sptr self*, *int m*)

null_source_sptr.**set_thread_priority**(*null_source_sptr self*, *int priority*) → int

null_source_sptr.**thread_priority**(*null_source_sptr self*) → int

gnuradio.blocks.**or_bb**(*size_t vlen=1*) → or_bb_sptr

output = input_0 | input_1 | , ... | input_N)

Bitwise boolean or across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

or_bb_sptr.**active_thread_priority**(*or_bb_sptr self*) → int

or_bb_sptr.**declare_sample_delay**(*or_bb_sptr self*, *int which*, *int delay*)
declare_sample_delay(or_bb_sptr self, unsigned int delay)

or_bb_sptr.**message_subscribers**(*or_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

or_bb_sptr.**min_noutput_items**(*or_bb_sptr self*) → int

or_bb_sptr.**pc_input_buffers_full_avg**(*or_bb_sptr self*, *int which*) → float
pc_input_buffers_full_avg(or_bb_sptr self) -> pmt_vector_float

or_bb_sptr.**pc_noutput_items_avg**(*or_bb_sptr self*) → float

or_bb_sptr.**pc_nproduced_avg**(*or_bb_sptr self*) → float

or_bb_sptr.**pc_output_buffers_full_avg**(*or_bb_sptr self*, *int which*) → float
pc_output_buffers_full_avg(or_bb_sptr self) -> pmt_vector_float

or_bb_sptr.**pc_throughput_avg**(*or_bb_sptr self*) → float

or_bb_sptr.**pc_work_time_avg**(*or_bb_sptr self*) → float

or_bb_sptr.**pc_work_time_total**(*or_bb_sptr self*) → float

or_bb_sptr.**sample_delay**(*or_bb_sptr self*, *int which*) → unsigned int

or_bb_sptr.**set_min_noutput_items**(*or_bb_sptr self*, *int m*)

or_bb_sptr.**set_thread_priority**(*or_bb_sptr self*, *int priority*) → int

or_bb_sptr.**thread_priority**(*or_bb_sptr self*) → int

gnuradio.blocks.**or_ii**(*size_t vlen=1*) → or_ii_sptr
output = input_0 | input_1 | , ... | input_N)

Bitwise boolean or across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

or_ii_sptr.**active_thread_priority**(*or_ii_sptr self*) → int

or_ii_sptr.**declare_sample_delay**(*or_ii_sptr self*, *int which*, *int delay*)
declare_sample_delay(or_ii_sptr self, unsigned int delay)

or_ii_sptr.**message_subscribers**(*or_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

or_ii_sptr.**min_noutput_items**(*or_ii_sptr self*) → int

or_ii_sptr.**pc_input_buffers_full_avg**(*or_ii_sptr self*, *int which*) → float
pc_input_buffers_full_avg(or_ii_sptr self) -> pmt_vector_float

or_ii_sptr.**pc_noutput_items_avg**(*or_ii_sptr self*) → float

or_ii_sptr.**pc_nproduced_avg**(*or_ii_sptr self*) → float

or_ii_sptr.**pc_output_buffers_full_avg**(*or_ii_sptr self*, *int which*) → float
pc_output_buffers_full_avg(or_ii_sptr self) -> pmt_vector_float

or_ii_sptr.**pc_throughput_avg**(*or_ii_sptr self*) → float

or_ii_sptr.**pc_work_time_avg**(*or_ii_sptr self*) → float

or_ii_sptr.**pc_work_time_total**(*or_ii_sptr self*) → float

or_ii_sptr.**sample_delay**(*or_ii_sptr self*, *int which*) → unsigned int

or_ii_sptr.**set_min_noutput_items**(*or_ii_sptr self*, *int m*)

or_ii_sptr.**set_thread_priority**(*or_ii_sptr self*, *int priority*) → int

or_ii_sptr.**thread_priority**(*or_ii_sptr self*) → int

gnuradio.blocks.**or_ss**(*size_t vlen=1*) → or_ss_sptr

output = input_0 | input_1 | , ... | input_N)

Bitwise boolean or across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

or_ss_sptr.**active_thread_priority**(*or_ss_sptr self*) → int

or_ss_sptr.**declare_sample_delay**(*or_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(or_ss_sptr self, unsigned int delay)

or_ss_sptr.**message_subscribers**(*or_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

or_ss_sptr.**min_noutput_items**(*or_ss_sptr self*) → int

or_ss_sptr.**pc_input_buffers_full_avg**(*or_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(or_ss_sptr self) -> pmt_vector_float

or_ss_sptr.**pc_noutput_items_avg**(*or_ss_sptr self*) → float

or_ss_sptr.**pc_nproduced_avg**(*or_ss_sptr self*) → float

or_ss_sptr.**pc_output_buffers_full_avg**(*or_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(or_ss_sptr self) -> pmt_vector_float

or_ss_sptr.**pc_throughput_avg**(*or_ss_sptr self*) → float

or_ss_sptr.**pc_work_time_avg**(*or_ss_sptr self*) → float

or_ss_sptr.**pc_work_time_total**(*or_ss_sptr self*) → float

or_ss_sptr.**sample_delay**(*or_ss_sptr self*, *int which*) → unsigned int

or_ss_sptr.**set_min_noutput_items**(*or_ss_sptr self*, *int m*)

or_ss_sptr.**set_thread_priority**(*or_ss_sptr self*, *int priority*) → int

or_ss_sptr.**thread_priority**(*or_ss_sptr self*) → int

gnuradio.blocks.**pack_k_bits_bb**(*unsigned int k*) → pack_k_bits_bb_sptr

Converts a stream of bytes with 1 bit in the LSB to a byte with k relevant bits.

This block takes in K bytes at a time, and uses the least significant bit to form a new byte.

Example: k = 4 in = [0,1,0,1, 0x81,0x00,0x00,0x00] out = [0x05, 0x08]

Constructor Specific Documentation:

Make a pack_k_bits block.

Parameters: **k** – number of bits to be packed.

pack_k_bits_bb_sptr.**active_thread_priority**(*pack_k_bits_bb_sptr self*) → int

pack_k_bits_bb_sptr.**declare_sample_delay**(*pack_k_bits_bb_sptr self*, *int which*, *int delay*)

declare_sample_delay(pack_k_bits_bb_sptr self, unsigned int delay)

pack_k_bits_bb_sptr.**message_subscribers**(*pack_k_bits_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pack_k_bits_bb_sptr.**min_noutput_items**(*pack_k_bits_bb_sptr self*) → int

pack_k_bits_bb_sptr.**pc_input_buffers_full_avg**(*pack_k_bits_bb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pack_k_bits_bb_sptr self) -> pmt_vector_float

pack_k_bits_bb_sptr.**pc_noutput_items_avg**(*pack_k_bits_bb_sptr self*) → float

pack_k_bits_bb_sptr.**pc_nproduced_avg**(*pack_k_bits_bb_sptr self*) → float

pack_k_bits_bb_sptr.**pc_output_buffers_full_avg**(*pack_k_bits_bb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pack_k_bits_bb_sptr self) -> pmt_vector_float

pack_k_bits_bb_sptr.**pc_throughput_avg**(*pack_k_bits_bb_sptr self*) → float

pack_k_bits_bb_sptr.**pc_work_time_avg**(*pack_k_bits_bb_sptr self*) → float

pack_k_bits_bb_sptr.**pc_work_time_total**(*pack_k_bits_bb_sptr self*) → float

pack_k_bits_bb_sptr.**sample_delay**(*pack_k_bits_bb_sptr self*, *int which*) → unsigned int

pack_k_bits_bb_sptr.**set_min_noutput_items**(*pack_k_bits_bb_sptr self*, *int m*)

pack_k_bits_bb_sptr.**set_thread_priority**(*pack_k_bits_bb_sptr self*, *int priority*) → int

pack_k_bits_bb_sptr.**thread_priority**(*pack_k_bits_bb_sptr self*) → int

gnuradio.blocks.**packed_to_unpacked_bb**(*unsigned int bits_per_chunk*, *gr::endianness_t endianness*) → packed_to_unpacked_bb_sptr

Convert a stream of packed bytes or shorts to stream of unpacked bytes or shorts.

input: stream of unsigned char; output: stream of unsigned char

This is the inverse of gr::blocks::unpacked_to_packed_XX.

The bits in the bytes or shorts input stream are grouped into chunks of bits and each resulting chunk is written right- justified to the output stream of bytes or shorts. All b or 16 bits of the each input bytes or short are processed. The right thing is done if bits_per_chunk is not a power of two.

The combination of gr::blocks::packed_to_unpacked_XX_ followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols.

Constructor Specific Documentation:

Parameters:
- **bits_per_chunk** –
- **endianness** –

packed_to_unpacked_bb_sptr.**active_thread_priority**(*packed_to_unpacked_bb_sptr self*) → int

packed_to_unpacked_bb_sptr.**declare_sample_delay**(*packed_to_unpacked_bb_sptr self*, *int which*, *int delay*)

declare_sample_delay(packed_to_unpacked_bb_sptr self, unsigned int delay)

packed_to_unpacked_bb_sptr.**message_subscribers**(*packed_to_unpacked_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

packed_to_unpacked_bb_sptr.**min_noutput_items**(*packed_to_unpacked_bb_sptr self*) → int

packed_to_unpacked_bb_sptr.**pc_input_buffers_full_avg**(*packed_to_unpacked_bb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(packed_to_unpacked_bb_sptr self) -> pmt_vector_float

packed_to_unpacked_bb_sptr.**pc_noutput_items_avg**(*packed_to_unpacked_bb_sptr self*) → float

packed_to_unpacked_bb_sptr.**pc_nproduced_avg**(*packed_to_unpacked_bb_sptr self*) → float

packed_to_unpacked_bb_sptr.**pc_output_buffers_full_avg**(*packed_to_unpacked_bb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(packed_to_unpacked_bb_sptr self) -> pmt_vector_float

packed_to_unpacked_bb_sptr.**pc_throughput_avg**(*packed_to_unpacked_bb_sptr self*) → float

packed_to_unpacked_bb_sptr.**pc_work_time_avg**(*packed_to_unpacked_bb_sptr self*) → float

packed_to_unpacked_bb_sptr.**pc_work_time_total**(*packed_to_unpacked_bb_sptr self*) → float

packed_to_unpacked_bb_sptr.**sample_delay**(*packed_to_unpacked_bb_sptr self*, *int which*) → unsigned int

packed_to_unpacked_bb_sptr.**set_min_noutput_items**(*packed_to_unpacked_bb_sptr self*, *int m*)

packed_to_unpacked_bb_sptr.**set_thread_priority**(*packed_to_unpacked_bb_sptr self*, *int priority*) → int

packed_to_unpacked_bb_sptr.**thread_priority**(*packed_to_unpacked_bb_sptr self*) → int

gnuradio.blocks.**packed_to_unpacked_ii**(*unsigned int bits_per_chunk*, *gr::endianness_t endianness*) → packed_to_unpacked_ii_sptr

Convert a stream of packed bytes or shorts to stream of unpacked bytes or shorts.

input: stream of int; output: stream of int

This is the inverse of gr::blocks::unpacked_to_packed_XX.

The bits in the bytes or shorts input stream are grouped into chunks of bits and each resulting chunk is written right- justified to the output stream of bytes or shorts. All b or 16 bits of the each input bytes or short are processed. The right thing is done if bits_per_chunk is not a power of two.

The combination of gr::blocks::packed_to_unpacked_XX_ followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols.

Constructor Specific Documentation:

Parameters:
- **bits_per_chunk** –
- **endianness** –

packed_to_unpacked_ii_sptr.**active_thread_priority**(*packed_to_unpacked_ii_sptr self*) → int

packed_to_unpacked_ii_sptr.**declare_sample_delay**(*packed_to_unpacked_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(packed_to_unpacked_ii_sptr self, unsigned int delay)

packed_to_unpacked_ii_sptr.**message_subscribers**(*packed_to_unpacked_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

packed_to_unpacked_ii_sptr.**min_noutput_items**(*packed_to_unpacked_ii_sptr self*) → int

packed_to_unpacked_ii_sptr.**pc_input_buffers_full_avg**(*packed_to_unpacked_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(packed_to_unpacked_ii_sptr self) -> pmt_vector_float

packed_to_unpacked_ii_sptr.**pc_noutput_items_avg**(*packed_to_unpacked_ii_sptr self*) → float

packed_to_unpacked_ii_sptr.**pc_nproduced_avg**(*packed_to_unpacked_ii_sptr self*) → float

packed_to_unpacked_ii_sptr.**pc_output_buffers_full_avg**(*packed_to_unpacked_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(packed_to_unpacked_ii_sptr self) -> pmt_vector_float

packed_to_unpacked_ii_sptr.**pc_throughput_avg**(*packed_to_unpacked_ii_sptr self*) → float

packed_to_unpacked_ii_sptr.**pc_work_time_avg**(*packed_to_unpacked_ii_sptr self*) → float

packed_to_unpacked_ii_sptr.**pc_work_time_total**(*packed_to_unpacked_ii_sptr self*) → float

packed_to_unpacked_ii_sptr.**sample_delay**(*packed_to_unpacked_ii_sptr self*, *int which*) → unsigned int

packed_to_unpacked_ii_sptr.**set_min_noutput_items**(*packed_to_unpacked_ii_sptr self*, *int m*)

packed_to_unpacked_ii_sptr.**set_thread_priority**(*packed_to_unpacked_ii_sptr self*, *int priority*) → int

packed_to_unpacked_ii_sptr.**thread_priority**(*packed_to_unpacked_ii_sptr self*) → int

gnuradio.blocks.**packed_to_unpacked_ss**(*unsigned int bits_per_chunk*, *gr::endianness_t endianness*) → packed_to_unpacked_ss_sptr

Convert a stream of packed bytes or shorts to stream of unpacked bytes or shorts.

input: stream of short; output: stream of short

This is the inverse of gr::blocks::unpacked_to_packed_XX.

The bits in the bytes or shorts input stream are grouped into chunks of bits and each resulting chunk is written right- justified to the output stream of bytes or shorts. All b or 16 bits of the each input bytes or short are processed. The right thing is done if bits_per_chunk is not a power of two.

The combination of gr::blocks::packed_to_unpacked_XX_ followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols.

Constructor Specific Documentation:

Parameters:
- **bits_per_chunk** –
- **endianness** –

packed_to_unpacked_ss_sptr.**active_thread_priority**(*packed_to_unpacked_ss_sptr self*) → int

packed_to_unpacked_ss_sptr.**declare_sample_delay**(*packed_to_unpacked_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(packed_to_unpacked_ss_sptr self, unsigned int delay)

packed_to_unpacked_ss_sptr.**message_subscribers**(*packed_to_unpacked_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

packed_to_unpacked_ss_sptr.**min_noutput_items**(*packed_to_unpacked_ss_sptr self*) → int

packed_to_unpacked_ss_sptr.**pc_input_buffers_full_avg**(*packed_to_unpacked_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(packed_to_unpacked_ss_sptr self) -> pmt_vector_float

packed_to_unpacked_ss_sptr.**pc_noutput_items_avg**(*packed_to_unpacked_ss_sptr self*) → float

packed_to_unpacked_ss_sptr.**pc_nproduced_avg**(*packed_to_unpacked_ss_sptr self*) → float

packed_to_unpacked_ss_sptr.**pc_output_buffers_full_avg**(*packed_to_unpacked_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(packed_to_unpacked_ss_sptr self) -> pmt_vector_float

packed_to_unpacked_ss_sptr.**pc_throughput_avg**(*packed_to_unpacked_ss_sptr self*) → float

packed_to_unpacked_ss_sptr.**pc_work_time_avg**(*packed_to_unpacked_ss_sptr self*) → float

packed_to_unpacked_ss_sptr.**pc_work_time_total**(*packed_to_unpacked_ss_sptr self*) → float

packed_to_unpacked_ss_sptr.**sample_delay**(*packed_to_unpacked_ss_sptr self*, *int which*) → unsigned int

packed_to_unpacked_ss_sptr.**set_min_noutput_items**(*packed_to_unpacked_ss_sptr self*, *int m*)

packed_to_unpacked_ss_sptr.**set_thread_priority**(*packed_to_unpacked_ss_sptr self*, *int priority*) → int

packed_to_unpacked_ss_sptr.**thread_priority**(*packed_to_unpacked_ss_sptr self*) → int

gnuradio.blocks.**patterned_interleaver**(*size_t itemsize*, *std::vector< int, std::allocator< int > > pattern*) → patterned_interleaver_sptr

Interleave items based on the provided vector .

Constructor Specific Documentation:

Make a patterned interleaver block.

Parameters:
- **itemsize** – stream itemsize
- **pattern** – vector that represents the interleaving pattern

patterned_interleaver_sptr.**active_thread_priority**(*patterned_interleaver_sptr self*) → int

patterned_interleaver_sptr.**declare_sample_delay**(*patterned_interleaver_sptr self*, *int which*, *int delay*)

declare_sample_delay(patterned_interleaver_sptr self, unsigned int delay)

patterned_interleaver_sptr.**message_subscribers**(*patterned_interleaver_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

patterned_interleaver_sptr.**min_noutput_items**(*patterned_interleaver_sptr self*) → int

patterned_interleaver_sptr.**pc_input_buffers_full_avg**(*patterned_interleaver_sptr self*, *int which*) → float

pc_input_buffers_full_avg(patterned_interleaver_sptr self) -> pmt_vector_float

patterned_interleaver_sptr.**pc_noutput_items_avg**(*patterned_interleaver_sptr self*) → float

patterned_interleaver_sptr.**pc_nproduced_avg**(*patterned_interleaver_sptr self*) → float

patterned_interleaver_sptr.**pc_output_buffers_full_avg**(*patterned_interleaver_sptr self*, *int which*) → float

pc_output_buffers_full_avg(patterned_interleaver_sptr self) -> pmt_vector_float

patterned_interleaver_sptr.**pc_throughput_avg**(*patterned_interleaver_sptr self*) → float

patterned_interleaver_sptr.**pc_work_time_avg**(*patterned_interleaver_sptr self*) → float

patterned_interleaver_sptr.**pc_work_time_total**(*patterned_interleaver_sptr self*) → float

patterned_interleaver_sptr.**sample_delay**(*patterned_interleaver_sptr self*, *int which*) → unsigned int

patterned_interleaver_sptr.**set_min_noutput_items**(*patterned_interleaver_sptr self*, *int m*)

patterned_interleaver_sptr.**set_thread_priority**(*patterned_interleaver_sptr self*, *int priority*) → int

patterned_interleaver_sptr.**thread_priority**(*patterned_interleaver_sptr self*) → int

gnuradio.blocks.**pdu_filter**(*swig_int_ptr k*, *swig_int_ptr v*, *bool invert=False*) → pdu_filter_sptr

Propagates only pdus containing k=>v in meta.

Constructor Specific Documentation:

Construct a PDU filter.

Parameters:
- **k** –
- **v** –
- **invert** –

pdu_filter_sptr.**active_thread_priority**(*pdu_filter_sptr self*) → int

pdu_filter_sptr.**declare_sample_delay**(*pdu_filter_sptr self*, *int which*, *int delay*)

declare_sample_delay(pdu_filter_sptr self, unsigned int delay)

pdu_filter_sptr.**message_subscribers**(*pdu_filter_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pdu_filter_sptr.**min_noutput_items**(*pdu_filter_sptr self*) → int

pdu_filter_sptr.**pc_input_buffers_full_avg**(*pdu_filter_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pdu_filter_sptr self) -> pmt_vector_float

pdu_filter_sptr.**pc_noutput_items_avg**(*pdu_filter_sptr self*) → float

pdu_filter_sptr.**pc_nproduced_avg**(*pdu_filter_sptr self*) → float

pdu_filter_sptr.**pc_output_buffers_full_avg**(*pdu_filter_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pdu_filter_sptr self) -> pmt_vector_float

pdu_filter_sptr.**pc_throughput_avg**(*pdu_filter_sptr self*) → float

pdu_filter_sptr.**pc_work_time_avg**(*pdu_filter_sptr self*) → float

pdu_filter_sptr.**pc_work_time_total**(*pdu_filter_sptr self*) → float

pdu_filter_sptr.**sample_delay**(*pdu_filter_sptr self*, *int which*) → unsigned int

pdu_filter_sptr.**set_inversion**(*pdu_filter_sptr self*, *bool invert*)

pdu_filter_sptr.**set_key**(*pdu_filter_sptr self*, *swig_int_ptr key*)

pdu_filter_sptr.**set_min_noutput_items**(*pdu_filter_sptr self*, *int m*)

pdu_filter_sptr.**set_thread_priority**(*pdu_filter_sptr self*, *int priority*) → int

pdu_filter_sptr.**set_val**(*pdu_filter_sptr self*, *swig_int_ptr val*)

pdu_filter_sptr.**thread_priority**(*pdu_filter_sptr self*) → int

gnuradio.blocks.**pdu_remove**(*swig_int_ptr k*) → pdu_remove_sptr

remove key k in pdu's meta field and pass on

Constructor Specific Documentation:

Construct a PDU meta remove block.

Parameters: **k** –

pdu_remove_sptr.**active_thread_priority**(*pdu_remove_sptr self*) → int

pdu_remove_sptr.**declare_sample_delay**(*pdu_remove_sptr self*, *int which*, *int delay*)

declare_sample_delay(pdu_remove_sptr self, unsigned int delay)

pdu_remove_sptr.**message_subscribers**(*pdu_remove_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pdu_remove_sptr.**min_noutput_items**(*pdu_remove_sptr self*) → int

pdu_remove_sptr.**pc_input_buffers_full_avg**(*pdu_remove_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pdu_remove_sptr self) -> pmt_vector_float

pdu_remove_sptr.**pc_noutput_items_avg**(*pdu_remove_sptr self*) → float

pdu_remove_sptr.**pc_nproduced_avg**(*pdu_remove_sptr self*) → float

pdu_remove_sptr.**pc_output_buffers_full_avg**(*pdu_remove_sptr self*, *int which*) → float
pc_output_buffers_full_avg(pdu_remove_sptr self) -> pmt_vector_float

pdu_remove_sptr.**pc_throughput_avg**(*pdu_remove_sptr self*) → float

pdu_remove_sptr.**pc_work_time_avg**(*pdu_remove_sptr self*) → float

pdu_remove_sptr.**pc_work_time_total**(*pdu_remove_sptr self*) → float

pdu_remove_sptr.**sample_delay**(*pdu_remove_sptr self*, *int which*) → unsigned int

pdu_remove_sptr.**set_key**(*pdu_remove_sptr self*, *swig_int_ptr key*)

pdu_remove_sptr.**set_min_noutput_items**(*pdu_remove_sptr self*, *int m*)

pdu_remove_sptr.**set_thread_priority**(*pdu_remove_sptr self*, *int priority*) → int

pdu_remove_sptr.**thread_priority**(*pdu_remove_sptr self*) → int

gnuradio.blocks.**pdu_set**(*swig_int_ptr k*, *swig_int_ptr v*) → pdu_set_sptr

Set k=>v in pdu's meta field and pass on.

Constructor Specific Documentation:

Construct a PDU meta set block.

Parameters:
- **k** –
- **v** –

pdu_set_sptr.**active_thread_priority**(*pdu_set_sptr self*) → int

pdu_set_sptr.**declare_sample_delay**(*pdu_set_sptr self*, *int which*, *int delay*)
declare_sample_delay(pdu_set_sptr self, unsigned int delay)

pdu_set_sptr.**message_subscribers**(*pdu_set_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pdu_set_sptr.**min_noutput_items**(*pdu_set_sptr self*) → int

pdu_set_sptr.**pc_input_buffers_full_avg**(*pdu_set_sptr self*, *int which*) → float
pc_input_buffers_full_avg(pdu_set_sptr self) -> pmt_vector_float

pdu_set_sptr.**pc_noutput_items_avg**(*pdu_set_sptr self*) → float

pdu_set_sptr.**pc_nproduced_avg**(*pdu_set_sptr self*) → float

pdu_set_sptr.**pc_output_buffers_full_avg**(*pdu_set_sptr self*, *int which*) → float
pc_output_buffers_full_avg(pdu_set_sptr self) -> pmt_vector_float

pdu_set_sptr.**pc_throughput_avg**(*pdu_set_sptr self*) → float

pdu_set_sptr.**pc_work_time_avg**(*pdu_set_sptr self*) → float

pdu_set_sptr.**pc_work_time_total**(*pdu_set_sptr self*) → float

pdu_set_sptr.**sample_delay**(*pdu_set_sptr self*, *int which*) → unsigned int

pdu_set_sptr.**set_key**(*pdu_set_sptr self*, *swig_int_ptr key*)

pdu_set_sptr.**set_min_noutput_items**(*pdu_set_sptr self*, *int m*)

pdu_set_sptr.**set_thread_priority**(*pdu_set_sptr self*, *int priority*) → int

pdu_set_sptr.**set_val**(*pdu_set_sptr self*, *swig_int_ptr val*)

pdu_set_sptr.**thread_priority**(*pdu_set_sptr self*) → int

gnuradio.blocks.**pdu_to_tagged_stream**(*gr::blocks::pdu::vector_type type*, *std::string const & lengthtagname*) → pdu_to_tagged_stream_sptr

Turns received PDUs into a tagged stream of items.

Constructor Specific Documentation:

Construct a pdu_to_tagged_stream block.

Parameters:
- **type** – PDU type of pdu::vector_type
- **lengthtagname** – The name of the tag that specifies how long the packet is. Defaults to 'packet_len'.

pdu_to_tagged_stream_sptr.**active_thread_priority**(*pdu_to_tagged_stream_sptr self*) → int

pdu_to_tagged_stream_sptr.**declare_sample_delay**(*pdu_to_tagged_stream_sptr self*, *int which*, *int delay*)

declare_sample_delay(pdu_to_tagged_stream_sptr self, unsigned int delay)

pdu_to_tagged_stream_sptr.**message_subscribers**(*pdu_to_tagged_stream_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

pdu_to_tagged_stream_sptr.**min_noutput_items**(*pdu_to_tagged_stream_sptr self*) → int

pdu_to_tagged_stream_sptr.**pc_input_buffers_full_avg**(*pdu_to_tagged_stream_sptr self*, *int which*) → float

pc_input_buffers_full_avg(pdu_to_tagged_stream_sptr self) -> pmt_vector_float

pdu_to_tagged_stream_sptr.**pc_noutput_items_avg**(*pdu_to_tagged_stream_sptr self*) → float

pdu_to_tagged_stream_sptr.**pc_nproduced_avg**(*pdu_to_tagged_stream_sptr self*) → float

pdu_to_tagged_stream_sptr.**pc_output_buffers_full_avg**(*pdu_to_tagged_stream_sptr self*, *int which*) → float

pc_output_buffers_full_avg(pdu_to_tagged_stream_sptr self) -> pmt_vector_float

pdu_to_tagged_stream_sptr.**pc_throughput_avg**(*pdu_to_tagged_stream_sptr self*) → float

pdu_to_tagged_stream_sptr.**pc_work_time_avg**(*pdu_to_tagged_stream_sptr self*) → float

pdu_to_tagged_stream_sptr.**pc_work_time_total**(*pdu_to_tagged_stream_sptr self*) → float

pdu_to_tagged_stream_sptr.**sample_delay**(*pdu_to_tagged_stream_sptr self*, *int which*) → unsigned int

pdu_to_tagged_stream_sptr.**set_min_noutput_items**(*pdu_to_tagged_stream_sptr self*, *int m*)

pdu_to_tagged_stream_sptr.**set_thread_priority**(*pdu_to_tagged_stream_sptr self*, *int priority*) → int

pdu_to_tagged_stream_sptr.**thread_priority**(*pdu_to_tagged_stream_sptr self*) → int

gnuradio.blocks.**peak_detector2_fb**(*float threshold_factor_rise=7*, *int look_ahead=1000*, *float alpha=0.001*) → peak_detector2_fb_sptr

Detect the peak of a signal.

If a peak is detected, this block outputs a 1, or it outputs 0's. A separate debug output may be connected, to view the internal estimated mean described below.

Constructor Specific Documentation:

Build a peak detector block with float in, byte out.

Parameters:
- **threshold_factor_rise** – The threshold factor determines when a peak is present. An average of the input signal is calculated (through a single-pole autoregressive filter) and when the value of the input signal goes over threshold_factor_rise*average, we assume we are in the neighborhood of a peak. The block will then find the position of the maximum within a window of look_ahead samples starting at the point where the threshold was crossed upwards.
- **look_ahead** – The look-ahead value is used when the threshold is crossed upwards to locate the peak within this range.
- **alpha** – One minus the pole of a single-pole autoregressive filter that evaluates the average of the input signal.

peak_detector2_fb_sptr.**active_thread_priority**(*peak_detector2_fb_sptr self*) → int

peak_detector2_fb_sptr.**alpha**(*peak_detector2_fb_sptr self*) → float

Get the alpha value of the running average.

peak_detector2_fb_sptr.**declare_sample_delay**(*peak_detector2_fb_sptr self*, *int which*, *int delay*)

declare_sample_delay(peak_detector2_fb_sptr self, unsigned int delay)

peak_detector2_fb_sptr.**look_ahead**(*peak_detector2_fb_sptr self*) → int

Get the look-ahead factor value.

peak_detector2_fb_sptr.**message_subscribers**(*peak_detector2_fb_sptr self*, *swig_int_ptr which_port*) →

swig_int_ptr

peak_detector2_fb_sptr.**min_noutput_items**(*peak_detector2_fb_sptr self*) → int

peak_detector2_fb_sptr.**pc_input_buffers_full_avg**(*peak_detector2_fb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(peak_detector2_fb_sptr self) -> pmt_vector_float

peak_detector2_fb_sptr.**pc_noutput_items_avg**(*peak_detector2_fb_sptr self*) → float

peak_detector2_fb_sptr.**pc_nproduced_avg**(*peak_detector2_fb_sptr self*) → float

peak_detector2_fb_sptr.**pc_output_buffers_full_avg**(*peak_detector2_fb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(peak_detector2_fb_sptr self) -> pmt_vector_float

peak_detector2_fb_sptr.**pc_throughput_avg**(*peak_detector2_fb_sptr self*) → float

peak_detector2_fb_sptr.**pc_work_time_avg**(*peak_detector2_fb_sptr self*) → float

peak_detector2_fb_sptr.**pc_work_time_total**(*peak_detector2_fb_sptr self*) → float

peak_detector2_fb_sptr.**sample_delay**(*peak_detector2_fb_sptr self*, *int which*) → unsigned int

peak_detector2_fb_sptr.**set_alpha**(*peak_detector2_fb_sptr self*, *float alpha*)

Set the running average alpha.

peak_detector2_fb_sptr.**set_look_ahead**(*peak_detector2_fb_sptr self*, *int look*)

Set the look-ahead factor.

peak_detector2_fb_sptr.**set_min_noutput_items**(*peak_detector2_fb_sptr self*, *int m*)

peak_detector2_fb_sptr.**set_thread_priority**(*peak_detector2_fb_sptr self*, *int priority*) → int

peak_detector2_fb_sptr.**set_threshold_factor_rise**(*peak_detector2_fb_sptr self*, *float thr*)

Set the threshold factor value for the rise time.

peak_detector2_fb_sptr.**thread_priority**(*peak_detector2_fb_sptr self*) → int

peak_detector2_fb_sptr.**threshold_factor_rise**(*peak_detector2_fb_sptr self*) → float

Get the threshold factor value for the rise time.

gnuradio.blocks.**peak_detector_fb**(*float threshold_factor_rise=0.25*, *float threshold_factor_fall=0.40*, *int look_ahead=10*, *float alpha=0.001*) → peak_detector_fb_sptr

Detect the peak of a signal.

If a peak is detected, this block outputs a 1, or it outputs 0's.

Constructor Specific Documentation:

Make a peak detector block.

Parameters:

- **threshold_factor_rise** – The threshold factor determines when a peak has started. An average of the signal is calculated and when the value of the signal goes over threshold_factor_rise*average, we start looking for a peak.
- **threshold_factor_fall** – The threshold factor determines when a peak has ended. An average of the signal is calculated and when the value of the signal goes below threshold_factor_fall*average, we stop looking for a peak.
- **look_ahead** – The look-ahead value is used when the threshold is found to look if there another peak within this step range. If there is a larger value, we set that as the peak and look ahead again. This is continued until the highest point is found with This look-ahead range.
- **alpha** – The gain value of a moving average filter

peak_detector_fb_sptr.**active_thread_priority**(*peak_detector_fb_sptr self*) → int

peak_detector_fb_sptr.**alpha**(*peak_detector_fb_sptr self*) → float

Get the alpha value of the running average.

peak_detector_fb_sptr.**declare_sample_delay**(*peak_detector_fb_sptr self*, *int which*, *int delay*)

declare_sample_delay(peak_detector_fb_sptr self, unsigned int delay)

peak_detector_fb_sptr.**look_ahead**(*peak_detector_fb_sptr self*) → int

Get the look-ahead factor value.

peak_detector_fb_sptr.**message_subscribers**(*peak_detector_fb_sptr self*, *swig_int_ptr which_port*) →

swig_int_ptr

peak_detector_fb_sptr.**min_noutput_items**(*peak_detector_fb_sptr self*) → int

peak_detector_fb_sptr.**pc_input_buffers_full_avg**(*peak_detector_fb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(peak_detector_fb_sptr self) -> pmt_vector_float

peak_detector_fb_sptr.**pc_noutput_items_avg**(*peak_detector_fb_sptr self*) → float

peak_detector_fb_sptr.**pc_nproduced_avg**(*peak_detector_fb_sptr self*) → float

peak_detector_fb_sptr.**pc_output_buffers_full_avg**(*peak_detector_fb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(peak_detector_fb_sptr self) -> pmt_vector_float

peak_detector_fb_sptr.**pc_throughput_avg**(*peak_detector_fb_sptr self*) → float

peak_detector_fb_sptr.**pc_work_time_avg**(*peak_detector_fb_sptr self*) → float

peak_detector_fb_sptr.**pc_work_time_total**(*peak_detector_fb_sptr self*) → float

peak_detector_fb_sptr.**sample_delay**(*peak_detector_fb_sptr self*, *int which*) → unsigned int

peak_detector_fb_sptr.**set_alpha**(*peak_detector_fb_sptr self*, *float alpha*)

Set the running average alpha.

peak_detector_fb_sptr.**set_look_ahead**(*peak_detector_fb_sptr self*, *int look*)

Set the look-ahead factor.

peak_detector_fb_sptr.**set_min_noutput_items**(*peak_detector_fb_sptr self*, *int m*)

peak_detector_fb_sptr.**set_thread_priority**(*peak_detector_fb_sptr self*, *int priority*) → int

peak_detector_fb_sptr.**set_threshold_factor_fall**(*peak_detector_fb_sptr self*, *float thr*)

Set the threshold factor value for the fall time.

peak_detector_fb_sptr.**set_threshold_factor_rise**(*peak_detector_fb_sptr self*, *float thr*)

Set the threshold factor value for the rise time.

peak_detector_fb_sptr.**thread_priority**(*peak_detector_fb_sptr self*) → int

peak_detector_fb_sptr.**threshold_factor_fall**(*peak_detector_fb_sptr self*) → float

Get the threshold factor value for the fall time.

peak_detector_fb_sptr.**threshold_factor_rise**(*peak_detector_fb_sptr self*) → float

Get the threshold factor value for the rise time.

gnuradio.blocks.**peak_detector_ib**(*float threshold_factor_rise=0.25*, *float threshold_factor_fall=0.40*, *int look_ahead=10*, *float alpha=0.001*) → peak_detector_ib_sptr

Detect the peak of a signal.

If a peak is detected, this block outputs a 1, or it outputs 0's.

Constructor Specific Documentation:

Make a peak detector block.

Parameters:

- **threshold_factor_rise** – The threshold factor determines when a peak has started. An average of the signal is calculated and when the value of the signal goes over threshold_factor_rise*average, we start looking for a peak.
- **threshold_factor_fall** – The threshold factor determines when a peak has ended. An average of the signal is calculated and when the value of the signal goes below threshold_factor_fall*average, we stop looking for a peak.
- **look_ahead** – The look-ahead value is used when the threshold is found to look if there another peak within this step range. If there is a larger value, we set that as the peak and look ahead again. This is continued until the highest point is found with This look-ahead range.
- **alpha** – The gain value of a moving average filter

peak_detector_ib_sptr.**active_thread_priority**(*peak_detector_ib_sptr self*) → int

peak_detector_ib_sptr.**alpha**(*peak_detector_ib_sptr self*) → float

Get the alpha value of the running average.

peak_detector_ib_sptr.**declare_sample_delay**(*peak_detector_ib_sptr self*, *int which*, *int delay*)

declare_sample_delay(peak_detector_ib_sptr self, unsigned int delay)

peak_detector_ib_sptr.**look_ahead**(*peak_detector_ib_sptr self*) → int

Get the look-ahead factor value.

peak_detector_ib_sptr.**message_subscribers**(*peak_detector_ib_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

peak_detector_ib_sptr.**min_noutput_items**(*peak_detector_ib_sptr self*) → int

peak_detector_ib_sptr.**pc_input_buffers_full_avg**(*peak_detector_ib_sptr self*, *int which*) → float

pc_input_buffers_full_avg(peak_detector_ib_sptr self) -> pmt_vector_float

peak_detector_ib_sptr.**pc_noutput_items_avg**(*peak_detector_ib_sptr self*) → float

peak_detector_ib_sptr.**pc_nproduced_avg**(*peak_detector_ib_sptr self*) → float

peak_detector_ib_sptr.**pc_output_buffers_full_avg**(*peak_detector_ib_sptr self*, *int which*) → float

pc_output_buffers_full_avg(peak_detector_ib_sptr self) -> pmt_vector_float

peak_detector_ib_sptr.**pc_throughput_avg**(*peak_detector_ib_sptr self*) → float

peak_detector_ib_sptr.**pc_work_time_avg**(*peak_detector_ib_sptr self*) → float

peak_detector_ib_sptr.**pc_work_time_total**(*peak_detector_ib_sptr self*) → float

peak_detector_ib_sptr.**sample_delay**(*peak_detector_ib_sptr self*, *int which*) → unsigned int

peak_detector_ib_sptr.**set_alpha**(*peak_detector_ib_sptr self*, *float alpha*)

Set the running average alpha.

peak_detector_ib_sptr.**set_look_ahead**(*peak_detector_ib_sptr self*, *int look*)

Set the look-ahead factor.

peak_detector_ib_sptr.**set_min_noutput_items**(*peak_detector_ib_sptr self*, *int m*)

peak_detector_ib_sptr.**set_thread_priority**(*peak_detector_ib_sptr self*, *int priority*) → int

peak_detector_ib_sptr.**set_threshold_factor_fall**(*peak_detector_ib_sptr self*, *float thr*)

Set the threshold factor value for the fall time.

peak_detector_ib_sptr.**set_threshold_factor_rise**(*peak_detector_ib_sptr self*, *float thr*)

Set the threshold factor value for the rise time.

peak_detector_ib_sptr.**thread_priority**(*peak_detector_ib_sptr self*) → int

peak_detector_ib_sptr.**threshold_factor_fall**(*peak_detector_ib_sptr self*) → float

Get the threshold factor value for the fall time.

peak_detector_ib_sptr.**threshold_factor_rise**(*peak_detector_ib_sptr self*) → float

Get the threshold factor value for the rise time.

gnuradio.blocks.**peak_detector_sb**(*float threshold_factor_rise=0.25*, *float threshold_factor_fall=0.40*, *int look_ahead=10*, *float alpha=0.001*) → peak_detector_sb_sptr

Detect the peak of a signal.

If a peak is detected, this block outputs a 1, or it outputs 0's.

Constructor Specific Documentation:

Make a peak detector block.

Parameters:

- **threshold_factor_rise** – The threshold factor determines when a peak has started. An average of the signal is calculated and when the value of the signal goes over threshold_factor_rise*average, we start looking for a peak.
- **threshold_factor_fall** – The threshold factor determines when a peak has ended. An average of the signal is calculated and when the value of the signal goes below threshold_factor_fall*average, we stop looking for a peak.
- **look_ahead** – The look-ahead value is used when the threshold is found to look if there another peak within this step range. If there is a larger value, we set that as the peak and look ahead again. This is continued until the highest point is found with This look-ahead range.
- **alpha** – The gain value of a moving average filter

peak_detector_sb_sptr.**active_thread_priority**(*peak_detector_sb_sptr self*) → int

peak_detector_sb_sptr.**alpha**(*peak_detector_sb_sptr self*) → float

Get the alpha value of the running average.

peak_detector_sb_sptr.**declare_sample_delay**(*peak_detector_sb_sptr self*, *int which*, *int delay*)

declare_sample_delay(peak_detector_sb_sptr self, unsigned int delay)

peak_detector_sb_sptr.**look_ahead**(*peak_detector_sb_sptr self*) → int

Get the look-ahead factor value.

peak_detector_sb_sptr.**message_subscribers**(*peak_detector_sb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

peak_detector_sb_sptr.**min_noutput_items**(*peak_detector_sb_sptr self*) → int

peak_detector_sb_sptr.**pc_input_buffers_full_avg**(*peak_detector_sb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(peak_detector_sb_sptr self) -> pmt_vector_float

peak_detector_sb_sptr.**pc_noutput_items_avg**(*peak_detector_sb_sptr self*) → float

peak_detector_sb_sptr.**pc_nproduced_avg**(*peak_detector_sb_sptr self*) → float

peak_detector_sb_sptr.**pc_output_buffers_full_avg**(*peak_detector_sb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(peak_detector_sb_sptr self) -> pmt_vector_float

peak_detector_sb_sptr.**pc_throughput_avg**(*peak_detector_sb_sptr self*) → float

peak_detector_sb_sptr.**pc_work_time_avg**(*peak_detector_sb_sptr self*) → float

peak_detector_sb_sptr.**pc_work_time_total**(*peak_detector_sb_sptr self*) → float

peak_detector_sb_sptr.**sample_delay**(*peak_detector_sb_sptr self*, *int which*) → unsigned int

peak_detector_sb_sptr.**set_alpha**(*peak_detector_sb_sptr self*, *float alpha*)

Set the running average alpha.

peak_detector_sb_sptr.**set_look_ahead**(*peak_detector_sb_sptr self*, *int look*)

Set the look-ahead factor.

peak_detector_sb_sptr.**set_min_noutput_items**(*peak_detector_sb_sptr self*, *int m*)

peak_detector_sb_sptr.**set_thread_priority**(*peak_detector_sb_sptr self*, *int priority*) → int

peak_detector_sb_sptr.**set_threshold_factor_fall**(*peak_detector_sb_sptr self*, *float thr*)

Set the threshold factor value for the fall time.

peak_detector_sb_sptr.**set_threshold_factor_rise**(*peak_detector_sb_sptr self*, *float thr*)

Set the threshold factor value for the rise time.

peak_detector_sb_sptr.**thread_priority**(*peak_detector_sb_sptr self*) → int

peak_detector_sb_sptr.**threshold_factor_fall**(*peak_detector_sb_sptr self*) → float

Get the threshold factor value for the fall time.

peak_detector_sb_sptr.**threshold_factor_rise**(*peak_detector_sb_sptr self*) → float

Get the threshold factor value for the rise time.

gnuradio.blocks.**plateau_detector_fb**(*int max_len*, *float threshold=0.9*) → plateau_detector_fb_sptr

Detects a plateau and marks the middle.

Detect a plateau of a-priori known height. Input is a stream of floats, the output is a stream of bytes. Whenever a plateau is detected, the middle of that plateau is marked with a '1' on the output stream (all other samples are left at zero).

You can use this in a Schmidl & Cox synchronisation algorithm to interpret the output of the normalized correlator. Just pass the length of the cyclic prefix (in samples) as the max_len parameter).

Unlike the peak detectors, you must the now the absolute height of the plateau. Whenever the amplitude exceeds the given threshold, it starts assuming the presence of a plateau.

An implicit hysteresis is provided by the fact that after detecting one plateau, it waits at least max_len samples before the next plateau can be detected.

Constructor Specific Documentation:

Parameters:
- **max_len** – Maximum length of the plateau
- **threshold** – Anything above this value is considered a plateau

plateau_detector_fb_sptr.**active_thread_priority**(*plateau_detector_fb_sptr self*) → int

plateau_detector_fb_sptr.**declare_sample_delay**(*plateau_detector_fb_sptr self*, *int which*, *int delay*)

declare_sample_delay(plateau_detector_fb_sptr self, unsigned int delay)

plateau_detector_fb_sptr.**message_subscribers**(*plateau_detector_fb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

plateau_detector_fb_sptr.**min_noutput_items**(*plateau_detector_fb_sptr self*) → int

plateau_detector_fb_sptr.**pc_input_buffers_full_avg**(*plateau_detector_fb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(plateau_detector_fb_sptr self) -> pmt_vector_float

plateau_detector_fb_sptr.**pc_noutput_items_avg**(*plateau_detector_fb_sptr self*) → float

plateau_detector_fb_sptr.**pc_nproduced_avg**(*plateau_detector_fb_sptr self*) → float

plateau_detector_fb_sptr.**pc_output_buffers_full_avg**(*plateau_detector_fb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(plateau_detector_fb_sptr self) -> pmt_vector_float

plateau_detector_fb_sptr.**pc_throughput_avg**(*plateau_detector_fb_sptr self*) → float

plateau_detector_fb_sptr.**pc_work_time_avg**(*plateau_detector_fb_sptr self*) → float

plateau_detector_fb_sptr.**pc_work_time_total**(*plateau_detector_fb_sptr self*) → float

plateau_detector_fb_sptr.**sample_delay**(*plateau_detector_fb_sptr self*, *int which*) → unsigned int

plateau_detector_fb_sptr.**set_min_noutput_items**(*plateau_detector_fb_sptr self*, *int m*)

plateau_detector_fb_sptr.**set_thread_priority**(*plateau_detector_fb_sptr self*, *int priority*) → int

plateau_detector_fb_sptr.**set_threshold**(*plateau_detector_fb_sptr self*, *float threshold*)

plateau_detector_fb_sptr.**thread_priority**(*plateau_detector_fb_sptr self*) → int

plateau_detector_fb_sptr.**threshold**(*plateau_detector_fb_sptr self*) → float

gnuradio.blocks.**probe_rate**(*size_t itemsize*, *double update_rate_ms=500.0*, *double alpha=0.0001*) → probe_rate_sptr

throughput measurement

Constructor Specific Documentation:

Make a throughput measurement block.

Parameters:
- **itemsize** – size of each stream item
- **update_rate_ms** – minimum update time in milliseconds
- **alpha** – gain for running average filter

probe_rate_sptr.**active_thread_priority**(*probe_rate_sptr self*) → int

probe_rate_sptr.**declare_sample_delay**(*probe_rate_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_rate_sptr self, unsigned int delay)

probe_rate_sptr.**message_subscribers**(*probe_rate_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_rate_sptr.**min_noutput_items**(*probe_rate_sptr self*) → int

probe_rate_sptr.**pc_input_buffers_full_avg**(*probe_rate_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_rate_sptr self) -> pmt_vector_float

probe_rate_sptr.**pc_noutput_items_avg**(*probe_rate_sptr self*) → float

probe_rate_sptr.**pc_nproduced_avg**(*probe_rate_sptr self*) → float

probe_rate_sptr.**pc_output_buffers_full_avg**(*probe_rate_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_rate_sptr self) -> pmt_vector_float

probe_rate_sptr.**pc_throughput_avg**(*probe_rate_sptr self*) → float

probe_rate_sptr.**pc_work_time_avg**(*probe_rate_sptr self*) → float

probe_rate_sptr.**pc_work_time_total**(*probe_rate_sptr self*) → float

probe_rate_sptr.**rate**(*probe_rate_sptr self*) → double

probe_rate_sptr.**sample_delay**(*probe_rate_sptr self*, *int which*) → unsigned int

probe_rate_sptr.**set_alpha**(*probe_rate_sptr self*, *double alpha*)

probe_rate_sptr.**set_min_noutput_items**(*probe_rate_sptr self*, *int m*)

probe_rate_sptr.**set_thread_priority**(*probe_rate_sptr self*, *int priority*) → int

probe_rate_sptr.**thread_priority**(*probe_rate_sptr self*) → int

gnuradio.blocks.**probe_signal_b**() → probe_signal_b_sptr

Sink that allows a sample to be grabbed from Python.

Constructor Specific Documentation:

probe_signal_b_sptr.**active_thread_priority**(*probe_signal_b_sptr self*) → int

probe_signal_b_sptr.**declare_sample_delay**(*probe_signal_b_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_b_sptr self, unsigned int delay)

probe_signal_b_sptr.**level**(*probe_signal_b_sptr self*) → unsigned char

probe_signal_b_sptr.**message_subscribers**(*probe_signal_b_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_b_sptr.**min_noutput_items**(*probe_signal_b_sptr self*) → int

probe_signal_b_sptr.**pc_input_buffers_full_avg**(*probe_signal_b_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_b_sptr self) -> pmt_vector_float

probe_signal_b_sptr.**pc_noutput_items_avg**(*probe_signal_b_sptr self*) → float

probe_signal_b_sptr.**pc_nproduced_avg**(*probe_signal_b_sptr self*) → float

probe_signal_b_sptr.**pc_output_buffers_full_avg**(*probe_signal_b_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_b_sptr self) -> pmt_vector_float

probe_signal_b_sptr.**pc_throughput_avg**(*probe_signal_b_sptr self*) → float

probe_signal_b_sptr.**pc_work_time_avg**(*probe_signal_b_sptr self*) → float

probe_signal_b_sptr.**pc_work_time_total**(*probe_signal_b_sptr self*) → float

probe_signal_b_sptr.**sample_delay**(*probe_signal_b_sptr self*, *int which*) → unsigned int

probe_signal_b_sptr.**set_min_noutput_items**(*probe_signal_b_sptr self*, *int m*)

probe_signal_b_sptr.**set_thread_priority**(*probe_signal_b_sptr self*, *int priority*) → int

probe_signal_b_sptr.**thread_priority**(*probe_signal_b_sptr self*) → int

gnuradio.blocks.**probe_signal_c**() → probe_signal_c_sptr

Sink that allows a sample to be grabbed from Python.

Constructor Specific Documentation:

probe_signal_c_sptr.**active_thread_priority**(*probe_signal_c_sptr self*) → int

probe_signal_c_sptr.**declare_sample_delay**(*probe_signal_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_c_sptr self, unsigned int delay)

probe_signal_c_sptr.**level**(*probe_signal_c_sptr self*) → gr_complex

probe_signal_c_sptr.**message_subscribers**(*probe_signal_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_c_sptr.**min_noutput_items**(*probe_signal_c_sptr self*) → int

probe_signal_c_sptr.**pc_input_buffers_full_avg**(*probe_signal_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_c_sptr self) -> pmt_vector_float

probe_signal_c_sptr.**pc_noutput_items_avg**(*probe_signal_c_sptr self*) → float

probe_signal_c_sptr.**pc_nproduced_avg**(*probe_signal_c_sptr self*) → float

probe_signal_c_sptr.**pc_output_buffers_full_avg**(*probe_signal_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_c_sptr self) -> pmt_vector_float

probe_signal_c_sptr.**pc_throughput_avg**(*probe_signal_c_sptr self*) → float

probe_signal_c_sptr.**pc_work_time_avg**(*probe_signal_c_sptr self*) → float

probe_signal_c_sptr.**pc_work_time_total**(*probe_signal_c_sptr self*) → float

probe_signal_c_sptr.**sample_delay**(*probe_signal_c_sptr self*, *int which*) → unsigned int

probe_signal_c_sptr.**set_min_noutput_items**(*probe_signal_c_sptr self*, *int m*)

probe_signal_c_sptr.**set_thread_priority**(*probe_signal_c_sptr self*, *int priority*) → int

probe_signal_c_sptr.**thread_priority**(*probe_signal_c_sptr self*) → int

gnuradio.blocks.**probe_signal_f**() → probe_signal_f_sptr

Sink that allows a sample to be grabbed from Python.

Constructor Specific Documentation:

probe_signal_f_sptr.**active_thread_priority**(*probe_signal_f_sptr self*) → int

probe_signal_f_sptr.**declare_sample_delay**(*probe_signal_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_f_sptr self, unsigned int delay)

probe_signal_f_sptr.**level**(*probe_signal_f_sptr self*) → float

probe_signal_f_sptr.**message_subscribers**(*probe_signal_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_f_sptr.**min_noutput_items**(*probe_signal_f_sptr self*) → int

probe_signal_f_sptr.**pc_input_buffers_full_avg**(*probe_signal_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_f_sptr self) -> pmt_vector_float

probe_signal_f_sptr.**pc_noutput_items_avg**(*probe_signal_f_sptr self*) → float

probe_signal_f_sptr.**pc_nproduced_avg**(*probe_signal_f_sptr self*) → float

probe_signal_f_sptr.**pc_output_buffers_full_avg**(*probe_signal_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_f_sptr self) -> pmt_vector_float

probe_signal_f_sptr.**pc_throughput_avg**(*probe_signal_f_sptr self*) → float

probe_signal_f_sptr.**pc_work_time_avg**(*probe_signal_f_sptr self*) → float

probe_signal_f_sptr.**pc_work_time_total**(*probe_signal_f_sptr self*) → float

probe_signal_f_sptr.**sample_delay**(*probe_signal_f_sptr self*, *int which*) → unsigned int

probe_signal_f_sptr.**set_min_noutput_items**(*probe_signal_f_sptr self*, *int m*)

probe_signal_f_sptr.**set_thread_priority**(*probe_signal_f_sptr self*, *int priority*) → int

probe_signal_f_sptr.**thread_priority**(*probe_signal_f_sptr self*) → int

gnuradio.blocks.**probe_signal_i**() → probe_signal_i_sptr

Sink that allows a sample to be grabbed from Python.

Constructor Specific Documentation:

probe_signal_i_sptr.**active_thread_priority**(*probe_signal_i_sptr self*) → int

probe_signal_i_sptr.**declare_sample_delay**(*probe_signal_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_i_sptr self, unsigned int delay)

probe_signal_i_sptr.**level**(*probe_signal_i_sptr self*) → int

probe_signal_i_sptr.**message_subscribers**(*probe_signal_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_i_sptr.**min_noutput_items**(*probe_signal_i_sptr self*) → int

probe_signal_i_sptr.**pc_input_buffers_full_avg**(*probe_signal_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_i_sptr self) -> pmt_vector_float

probe_signal_i_sptr.**pc_noutput_items_avg**(*probe_signal_i_sptr self*) → float

probe_signal_i_sptr.**pc_nproduced_avg**(*probe_signal_i_sptr self*) → float

probe_signal_i_sptr.**pc_output_buffers_full_avg**(*probe_signal_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_i_sptr self) -> pmt_vector_float

probe_signal_i_sptr.**pc_throughput_avg**(*probe_signal_i_sptr self*) → float

probe_signal_i_sptr.**pc_work_time_avg**(*probe_signal_i_sptr self*) → float

probe_signal_i_sptr.**pc_work_time_total**(*probe_signal_i_sptr self*) → float

probe_signal_i_sptr.**sample_delay**(*probe_signal_i_sptr self*, *int which*) → unsigned int

probe_signal_i_sptr.**set_min_noutput_items**(*probe_signal_i_sptr self*, *int m*)

probe_signal_i_sptr.**set_thread_priority**(*probe_signal_i_sptr self*, *int priority*) → int

probe_signal_i_sptr.**thread_priority**(*probe_signal_i_sptr self*) → int

gnuradio.blocks.**probe_signal_s**() → probe_signal_s_sptr

Sink that allows a sample to be grabbed from Python.

Constructor Specific Documentation:

probe_signal_s_sptr.**active_thread_priority**(*probe_signal_s_sptr self*) → int

probe_signal_s_sptr.**declare_sample_delay**(*probe_signal_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_s_sptr self, unsigned int delay)

probe_signal_s_sptr.**level**(*probe_signal_s_sptr self*) → short

probe_signal_s_sptr.**message_subscribers**(*probe_signal_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_s_sptr.**min_noutput_items**(*probe_signal_s_sptr self*) → int

probe_signal_s_sptr.**pc_input_buffers_full_avg**(*probe_signal_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_s_sptr self) -> pmt_vector_float

probe_signal_s_sptr.**pc_noutput_items_avg**(*probe_signal_s_sptr self*) → float

probe_signal_s_sptr.**pc_nproduced_avg**(*probe_signal_s_sptr self*) → float

probe_signal_s_sptr.**pc_output_buffers_full_avg**(*probe_signal_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_s_sptr self) -> pmt_vector_float

probe_signal_s_sptr.**pc_throughput_avg**(*probe_signal_s_sptr self*) → float

probe_signal_s_sptr.**pc_work_time_avg**(*probe_signal_s_sptr self*) → float

probe_signal_s_sptr.**pc_work_time_total**(*probe_signal_s_sptr self*) → float

probe_signal_s_sptr.**sample_delay**(*probe_signal_s_sptr self*, *int which*) → unsigned int

probe_signal_s_sptr.**set_min_noutput_items**(*probe_signal_s_sptr self*, *int m*)

probe_signal_s_sptr.**set_thread_priority**(*probe_signal_s_sptr self*, *int priority*) → int

probe_signal_s_sptr.**thread_priority**(*probe_signal_s_sptr self*) → int

gnuradio.blocks.**probe_signal_vb**(*size_t size*) → probe_signal_vb_sptr

Sink that allows a vector of samples to be grabbed from Python.

Constructor Specific Documentation:

Parameters: **size** –

probe_signal_vb_sptr.**active_thread_priority**(*probe_signal_vb_sptr self*) → int

probe_signal_vb_sptr.**declare_sample_delay**(*probe_signal_vb_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_vb_sptr self, unsigned int delay)

probe_signal_vb_sptr.**level**(*probe_signal_vb_sptr self*) → std::vector< unsigned char,std::allocator< unsigned char > >

probe_signal_vb_sptr.**message_subscribers**(*probe_signal_vb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_vb_sptr.**min_noutput_items**(*probe_signal_vb_sptr self*) → int

probe_signal_vb_sptr.**pc_input_buffers_full_avg**(*probe_signal_vb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_vb_sptr self) -> pmt_vector_float

probe_signal_vb_sptr.**pc_noutput_items_avg**(*probe_signal_vb_sptr self*) → float

probe_signal_vb_sptr.**pc_nproduced_avg**(*probe_signal_vb_sptr self*) → float

probe_signal_vb_sptr.**pc_output_buffers_full_avg**(*probe_signal_vb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_vb_sptr self) -> pmt_vector_float

probe_signal_vb_sptr.**pc_throughput_avg**(*probe_signal_vb_sptr self*) → float

probe_signal_vb_sptr.**pc_work_time_avg**(*probe_signal_vb_sptr self*) → float

probe_signal_vb_sptr.**pc_work_time_total**(*probe_signal_vb_sptr self*) → float

probe_signal_vb_sptr.**sample_delay**(*probe_signal_vb_sptr self*, *int which*) → unsigned int

probe_signal_vb_sptr.**set_min_noutput_items**(*probe_signal_vb_sptr self*, *int m*)

probe_signal_vb_sptr.**set_thread_priority**(*probe_signal_vb_sptr self*, *int priority*) → int

probe_signal_vb_sptr.**thread_priority**(*probe_signal_vb_sptr self*) → int

gnuradio.blocks.**probe_signal_vc**(*size_t size*) → probe_signal_vc_sptr

Sink that allows a vector of samples to be grabbed from Python.

Constructor Specific Documentation:

Parameters: **size** –

probe_signal_vc_sptr.**active_thread_priority**(*probe_signal_vc_sptr self*) → int

probe_signal_vc_sptr.**declare_sample_delay**(*probe_signal_vc_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_vc_sptr self, unsigned int delay)

probe_signal_vc_sptr.**level**(*probe_signal_vc_sptr self*) → pmt_vector_cfloat

probe_signal_vc_sptr.**message_subscribers**(*probe_signal_vc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_vc_sptr.**min_noutput_items**(*probe_signal_vc_sptr self*) → int

probe_signal_vc_sptr.**pc_input_buffers_full_avg**(*probe_signal_vc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_vc_sptr self) -> pmt_vector_float

probe_signal_vc_sptr.**pc_noutput_items_avg**(*probe_signal_vc_sptr self*) → float

probe_signal_vc_sptr.**pc_nproduced_avg**(*probe_signal_vc_sptr self*) → float

probe_signal_vc_sptr.**pc_output_buffers_full_avg**(*probe_signal_vc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_vc_sptr self) -> pmt_vector_float

probe_signal_vc_sptr.**pc_throughput_avg**(*probe_signal_vc_sptr self*) → float

probe_signal_vc_sptr.**pc_work_time_avg**(*probe_signal_vc_sptr self*) → float

probe_signal_vc_sptr.**pc_work_time_total**(*probe_signal_vc_sptr self*) → float

probe_signal_vc_sptr.**sample_delay**(*probe_signal_vc_sptr self*, *int which*) → unsigned int

probe_signal_vc_sptr.**set_min_noutput_items**(*probe_signal_vc_sptr self*, *int m*)

probe_signal_vc_sptr.**set_thread_priority**(*probe_signal_vc_sptr self*, *int priority*) → int

probe_signal_vc_sptr.**thread_priority**(*probe_signal_vc_sptr self*) → int

gnuradio.blocks.**probe_signal_vf**(*size_t size*) → probe_signal_vf_sptr

Sink that allows a vector of samples to be grabbed from Python.

Constructor Specific Documentation:

Parameters: **size** –

probe_signal_vf_sptr.**active_thread_priority**(*probe_signal_vf_sptr self*) → int

probe_signal_vf_sptr.**declare_sample_delay**(*probe_signal_vf_sptr self*, *int which*, *int delay*)
declare_sample_delay(probe_signal_vf_sptr self, unsigned int delay)

probe_signal_vf_sptr.**level**(*probe_signal_vf_sptr self*) → pmt_vector_float

probe_signal_vf_sptr.**message_subscribers**(*probe_signal_vf_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_vf_sptr.**min_noutput_items**(*probe_signal_vf_sptr self*) → int

probe_signal_vf_sptr.**pc_input_buffers_full_avg**(*probe_signal_vf_sptr self*, *int which*) → float
pc_input_buffers_full_avg(probe_signal_vf_sptr self) -> pmt_vector_float

probe_signal_vf_sptr.**pc_noutput_items_avg**(*probe_signal_vf_sptr self*) → float

probe_signal_vf_sptr.**pc_nproduced_avg**(*probe_signal_vf_sptr self*) → float

probe_signal_vf_sptr.**pc_output_buffers_full_avg**(*probe_signal_vf_sptr self*, *int which*) → float
pc_output_buffers_full_avg(probe_signal_vf_sptr self) -> pmt_vector_float

probe_signal_vf_sptr.**pc_throughput_avg**(*probe_signal_vf_sptr self*) → float

probe_signal_vf_sptr.**pc_work_time_avg**(*probe_signal_vf_sptr self*) → float

probe_signal_vf_sptr.**pc_work_time_total**(*probe_signal_vf_sptr self*) → float

probe_signal_vf_sptr.**sample_delay**(*probe_signal_vf_sptr self*, *int which*) → unsigned int

probe_signal_vf_sptr.**set_min_noutput_items**(*probe_signal_vf_sptr self*, *int m*)

probe_signal_vf_sptr.**set_thread_priority**(*probe_signal_vf_sptr self*, *int priority*) → int

probe_signal_vf_sptr.**thread_priority**(*probe_signal_vf_sptr self*) → int

gnuradio.blocks.**probe_signal_vi**(*size_t size*) → probe_signal_vi_sptr
Sink that allows a vector of samples to be grabbed from Python.

Constructor Specific Documentation:

Parameters: **size** –

probe_signal_vi_sptr.**active_thread_priority**(*probe_signal_vi_sptr self*) → int

probe_signal_vi_sptr.**declare_sample_delay**(*probe_signal_vi_sptr self*, *int which*, *int delay*)
declare_sample_delay(probe_signal_vi_sptr self, unsigned int delay)

probe_signal_vi_sptr.**level**(*probe_signal_vi_sptr self*) → std::vector< int,std::allocator< int > >

probe_signal_vi_sptr.**message_subscribers**(*probe_signal_vi_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_vi_sptr.**min_noutput_items**(*probe_signal_vi_sptr self*) → int

probe_signal_vi_sptr.**pc_input_buffers_full_avg**(*probe_signal_vi_sptr self*, *int which*) → float
pc_input_buffers_full_avg(probe_signal_vi_sptr self) -> pmt_vector_float

probe_signal_vi_sptr.**pc_noutput_items_avg**(*probe_signal_vi_sptr self*) → float

probe_signal_vi_sptr.**pc_nproduced_avg**(*probe_signal_vi_sptr self*) → float

probe_signal_vi_sptr.**pc_output_buffers_full_avg**(*probe_signal_vi_sptr self*, *int which*) → float
pc_output_buffers_full_avg(probe_signal_vi_sptr self) -> pmt_vector_float

probe_signal_vi_sptr.**pc_throughput_avg**(*probe_signal_vi_sptr self*) → float

probe_signal_vi_sptr.**pc_work_time_avg**(*probe_signal_vi_sptr self*) → float

probe_signal_vi_sptr.**pc_work_time_total**(*probe_signal_vi_sptr self*) → float

probe_signal_vi_sptr.**sample_delay**(*probe_signal_vi_sptr self*, *int which*) → unsigned int

probe_signal_vi_sptr.**set_min_noutput_items**(*probe_signal_vi_sptr self*, *int m*)

probe_signal_vi_sptr.**set_thread_priority**(*probe_signal_vi_sptr self*, *int priority*) → int

probe_signal_vi_sptr.**thread_priority**(*probe_signal_vi_sptr self*) → int

gnuradio.blocks.**probe_signal_vs**(*size_t size*) → probe_signal_vs_sptr

Sink that allows a vector of samples to be grabbed from Python.

Constructor Specific Documentation:

Parameters: **size** –

probe_signal_vs_sptr.**active_thread_priority**(*probe_signal_vs_sptr self*) → int

probe_signal_vs_sptr.**declare_sample_delay**(*probe_signal_vs_sptr self*, *int which*, *int delay*)

declare_sample_delay(probe_signal_vs_sptr self, unsigned int delay)

probe_signal_vs_sptr.**level**(*probe_signal_vs_sptr self*) → std::vector< short,std::allocator< short > >

probe_signal_vs_sptr.**message_subscribers**(*probe_signal_vs_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

probe_signal_vs_sptr.**min_noutput_items**(*probe_signal_vs_sptr self*) → int

probe_signal_vs_sptr.**pc_input_buffers_full_avg**(*probe_signal_vs_sptr self*, *int which*) → float

pc_input_buffers_full_avg(probe_signal_vs_sptr self) -> pmt_vector_float

probe_signal_vs_sptr.**pc_noutput_items_avg**(*probe_signal_vs_sptr self*) → float

probe_signal_vs_sptr.**pc_nproduced_avg**(*probe_signal_vs_sptr self*) → float

probe_signal_vs_sptr.**pc_output_buffers_full_avg**(*probe_signal_vs_sptr self*, *int which*) → float

pc_output_buffers_full_avg(probe_signal_vs_sptr self) -> pmt_vector_float

probe_signal_vs_sptr.**pc_throughput_avg**(*probe_signal_vs_sptr self*) → float

probe_signal_vs_sptr.**pc_work_time_avg**(*probe_signal_vs_sptr self*) → float

probe_signal_vs_sptr.**pc_work_time_total**(*probe_signal_vs_sptr self*) → float

probe_signal_vs_sptr.**sample_delay**(*probe_signal_vs_sptr self*, *int which*) → unsigned int

probe_signal_vs_sptr.**set_min_noutput_items**(*probe_signal_vs_sptr self*, *int m*)

probe_signal_vs_sptr.**set_thread_priority**(*probe_signal_vs_sptr self*, *int priority*) → int

probe_signal_vs_sptr.**thread_priority**(*probe_signal_vs_sptr self*) → int

gnuradio.blocks.**random_pdu**(*int mintime*, *int maxtime*, *char byte_mask=0xFF*, *int length_modulo=1*) → random_pdu_sptr

Sends a random PDU at intervals.

Constructor Specific Documentation:

Construct a random PDU generator.

Parameters:
- **mintime** –
- **maxtime** –
- **byte_mask** –
- **length_modulo** –

random_pdu_sptr.**active_thread_priority**(*random_pdu_sptr self*) → int

random_pdu_sptr.**declare_sample_delay**(*random_pdu_sptr self*, *int which*, *int delay*)

declare_sample_delay(random_pdu_sptr self, unsigned int delay)

random_pdu_sptr.**message_subscribers**(*random_pdu_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

random_pdu_sptr.**min_noutput_items**(*random_pdu_sptr self*) → int

random_pdu_sptr.**pc_input_buffers_full_avg**(*random_pdu_sptr self*, *int which*) → float

pc_input_buffers_full_avg(random_pdu_sptr self) -> pmt_vector_float

random_pdu_sptr.**pc_noutput_items_avg**(*random_pdu_sptr self*) → float

random_pdu_sptr.**pc_nproduced_avg**(*random_pdu_sptr self*) → float

random_pdu_sptr.**pc_output_buffers_full_avg**(*random_pdu_sptr self*, *int which*) → float

pc_output_buffers_full_avg(random_pdu_sptr self) -> pmt_vector_float

random_pdu_sptr.**pc_throughput_avg**(*random_pdu_sptr self*) → float

random_pdu_sptr.**pc_work_time_avg**(*random_pdu_sptr self*) → float

random_pdu_sptr.**pc_work_time_total**(*random_pdu_sptr self*) → float

random_pdu_sptr.**sample_delay**(*random_pdu_sptr self*, *int which*) → unsigned int

random_pdu_sptr.**set_min_noutput_items**(*random_pdu_sptr self*, *int m*)

random_pdu_sptr.**set_thread_priority**(*random_pdu_sptr self*, *int priority*) → int

random_pdu_sptr.**thread_priority**(*random_pdu_sptr self*) → int

gnuradio.blocks.**regenerate_bb**(*int period*, *unsigned int max_regen=500*) → regenerate_bb_sptr

Detect the peak of a signal and repeat every period samples.

If a peak is detected, this block outputs a 1 repeated every period samples until reset by detection of another 1 on the input or stopped after max_regen regenerations have occurred.

Note that if max_regen=(-1)/ULONG_MAX then the regeneration will run forever.

Constructor Specific Documentation:

Make a regenerate block.

Parameters:
- **period** – The number of samples between regenerations
- **max_regen** – The maximum number of regenerations to perform; if set to ULONG_MAX, it will regenerate continuously.

regenerate_bb_sptr.**active_thread_priority**(*regenerate_bb_sptr self*) → int

regenerate_bb_sptr.**declare_sample_delay**(*regenerate_bb_sptr self*, *int which*, *int delay*)

declare_sample_delay(regenerate_bb_sptr self, unsigned int delay)

regenerate_bb_sptr.**max_regen**(*regenerate_bb_sptr self*) → unsigned int

return the maximum regeneration count.

regenerate_bb_sptr.**message_subscribers**(*regenerate_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

regenerate_bb_sptr.**min_noutput_items**(*regenerate_bb_sptr self*) → int

regenerate_bb_sptr.**pc_input_buffers_full_avg**(*regenerate_bb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(regenerate_bb_sptr self) -> pmt_vector_float

regenerate_bb_sptr.**pc_noutput_items_avg**(*regenerate_bb_sptr self*) → float

regenerate_bb_sptr.**pc_nproduced_avg**(*regenerate_bb_sptr self*) → float

regenerate_bb_sptr.**pc_output_buffers_full_avg**(*regenerate_bb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(regenerate_bb_sptr self) -> pmt_vector_float

regenerate_bb_sptr.**pc_throughput_avg**(*regenerate_bb_sptr self*) → float

regenerate_bb_sptr.**pc_work_time_avg**(*regenerate_bb_sptr self*) → float

regenerate_bb_sptr.**pc_work_time_total**(*regenerate_bb_sptr self*) → float

regenerate_bb_sptr.**period**(*regenerate_bb_sptr self*) → int

return the regeneration period.

regenerate_bb_sptr.**sample_delay**(*regenerate_bb_sptr self*, *int which*) → unsigned int

regenerate_bb_sptr.**set_max_regen**(*regenerate_bb_sptr self*, *unsigned int regen*)

Reset the maximum regeneration count; this will reset the current regen.

regenerate_bb_sptr.**set_min_noutput_items**(*regenerate_bb_sptr self*, *int m*)

regenerate_bb_sptr.**set_period**(*regenerate_bb_sptr self*, *int period*)

Reset the period of regenerations; this will reset the current regen.

regenerate_bb_sptr.**set_thread_priority**(*regenerate_bb_sptr self*, *int priority*) → int

regenerate_bb_sptr.**thread_priority**(*regenerate_bb_sptr self*) → int

gnuradio.blocks.**repack_bits_bb**(*int k*, *int l=8*, *std::string const & tsb_tag_key*, *bool align_output=False*, *gr::endianness_t endianness*) → repack_bits_bb_sptr

Repack bits from the input stream onto bits of the output stream.

No bits are lost here; any value for k and l (within [1, 8]) is allowed. On every fresh input byte, it starts reading on the LSB, and starts copying to the LSB as well.

When supplying a tag name, this block operates on tagged streams. In this case, it can happen that the input data or the output data becomes unaligned when k * input length is not equal to l * output length. In this case, the parameter is used to decide which data packet to align.

Usually, is false for unpacking (k=8, l < 8) and false for reversing that.

Example Say you're tx'ing 8-PSK and therefore set k=8, l=3 on the transmit side before the modulator. Now assume you're transmitting a single byte of data. Your incoming tagged stream has length 1, the outgoing has length 3. However, the third item is actually only carrying 2 bits of relevant data, the bits do not align with the boundaries. So you set = false, because the output can be unaligned.

Now say you're doing the inverse: packing those three items into full bytes. How do you interpret those three bytes? Without this flag, you'd have to assume there's 9 relevant bits in there, so you'd end up with 2 bytes of output data. But in the packing case, you want the to be aligned; all output bits must be useful. By asserting this flag, the packing algorithm tries to do this and in this case assumes that since we have alignment after 8 bits, the 9th can be discarded.

Constructor Specific Documentation:

Parameters:
- **k** – Number of relevant bits on the input stream
- **l** – Number of relevant bits on the output stream
- **tsb_tag_key** – If not empty, this is the key for the length tag.
- **align_output** – If tsb_tag_key is given, this controls if the input or the output is aligned.
- **endianness** – The endianness of the output data stream (LSB or MSB).

repack_bits_bb_sptr.**active_thread_priority**(*repack_bits_bb_sptr self*) → int

repack_bits_bb_sptr.**declare_sample_delay**(*repack_bits_bb_sptr self*, *int which*, *int delay*)
declare_sample_delay(repack_bits_bb_sptr self, unsigned int delay)

repack_bits_bb_sptr.**message_subscribers**(*repack_bits_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

repack_bits_bb_sptr.**min_noutput_items**(*repack_bits_bb_sptr self*) → int

repack_bits_bb_sptr.**pc_input_buffers_full_avg**(*repack_bits_bb_sptr self*, *int which*) → float
pc_input_buffers_full_avg(repack_bits_bb_sptr self) -> pmt_vector_float

repack_bits_bb_sptr.**pc_noutput_items_avg**(*repack_bits_bb_sptr self*) → float

repack_bits_bb_sptr.**pc_nproduced_avg**(*repack_bits_bb_sptr self*) → float

repack_bits_bb_sptr.**pc_output_buffers_full_avg**(*repack_bits_bb_sptr self*, *int which*) → float
pc_output_buffers_full_avg(repack_bits_bb_sptr self) -> pmt_vector_float

repack_bits_bb_sptr.**pc_throughput_avg**(*repack_bits_bb_sptr self*) → float

repack_bits_bb_sptr.**pc_work_time_avg**(*repack_bits_bb_sptr self*) → float

repack_bits_bb_sptr.**pc_work_time_total**(*repack_bits_bb_sptr self*) → float

repack_bits_bb_sptr.**sample_delay**(*repack_bits_bb_sptr self*, *int which*) → unsigned int

repack_bits_bb_sptr.**set_k_and_l**(*repack_bits_bb_sptr self*, *int k*, *int l*)

repack_bits_bb_sptr.**set_min_noutput_items**(*repack_bits_bb_sptr self*, *int m*)

repack_bits_bb_sptr.**set_thread_priority**(*repack_bits_bb_sptr self*, *int priority*) → int

repack_bits_bb_sptr.**thread_priority**(*repack_bits_bb_sptr self*) → int

gnuradio.blocks.**repeat**(*size_t itemsize*, *int repeat*) → repeat_sptr
repeat each input times

Message Ports:

Constructor Specific Documentation:

Make a repeat block.

Parameters:
- **itemsize** – stream itemsize
- **repeat** – number of times to repeat the input

repeat_sptr.**active_thread_priority**(*repeat_sptr self*) → int

repeat_sptr.**declare_sample_delay**(*repeat_sptr self*, *int which*, *int delay*)

declare_sample_delay(repeat_sptr self, unsigned int delay)

repeat_sptr.**interpolation**(*repeat_sptr self*) → int

Return current interpolation.

repeat_sptr.**message_subscribers**(*repeat_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

repeat_sptr.**min_noutput_items**(*repeat_sptr self*) → int

repeat_sptr.**pc_input_buffers_full_avg**(*repeat_sptr self*, *int which*) → float

pc_input_buffers_full_avg(repeat_sptr self) -> pmt_vector_float

repeat_sptr.**pc_noutput_items_avg**(*repeat_sptr self*) → float

repeat_sptr.**pc_nproduced_avg**(*repeat_sptr self*) → float

repeat_sptr.**pc_output_buffers_full_avg**(*repeat_sptr self*, *int which*) → float

pc_output_buffers_full_avg(repeat_sptr self) -> pmt_vector_float

repeat_sptr.**pc_throughput_avg**(*repeat_sptr self*) → float

repeat_sptr.**pc_work_time_avg**(*repeat_sptr self*) → float

repeat_sptr.**pc_work_time_total**(*repeat_sptr self*) → float

repeat_sptr.**sample_delay**(*repeat_sptr self*, *int which*) → unsigned int

repeat_sptr.**set_interpolation**(*repeat_sptr self*, *int interp*)

sets the interpolation

Call this method in a callback to adjust the interpolation at run time.

repeat_sptr.**set_min_noutput_items**(*repeat_sptr self*, *int m*)

repeat_sptr.**set_thread_priority**(*repeat_sptr self*, *int priority*) → int

repeat_sptr.**thread_priority**(*repeat_sptr self*) → int

gnuradio.blocks.**rms_cf**(*double alpha=0.0001*) → rms_cf_sptr

RMS average power.

Constructor Specific Documentation:

Make an RMS calc. block.

Parameters: **alpha** – gain for running average filter.

rms_cf_sptr.**active_thread_priority**(*rms_cf_sptr self*) → int

rms_cf_sptr.**declare_sample_delay**(*rms_cf_sptr self*, *int which*, *int delay*)

declare_sample_delay(rms_cf_sptr self, unsigned int delay)

rms_cf_sptr.**message_subscribers**(*rms_cf_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

rms_cf_sptr.**min_noutput_items**(*rms_cf_sptr self*) → int

rms_cf_sptr.**pc_input_buffers_full_avg**(*rms_cf_sptr self*, *int which*) → float

pc_input_buffers_full_avg(rms_cf_sptr self) -> pmt_vector_float

rms_cf_sptr.**pc_noutput_items_avg**(*rms_cf_sptr self*) → float

rms_cf_sptr.**pc_nproduced_avg**(*rms_cf_sptr self*) → float

rms_cf_sptr.**pc_output_buffers_full_avg**(*rms_cf_sptr self*, *int which*) → float

pc_output_buffers_full_avg(rms_cf_sptr self) -> pmt_vector_float

rms_cf_sptr.**pc_throughput_avg**(*rms_cf_sptr self*) → float

rms_cf_sptr.**pc_work_time_avg**(*rms_cf_sptr self*) → float

rms_cf_sptr.**pc_work_time_total**(*rms_cf_sptr self*) → float

rms_cf_sptr.**sample_delay**(*rms_cf_sptr self*, *int which*) → unsigned int

rms_cf_sptr.**set_alpha**(*rms_cf_sptr self*, *double alpha*)

rms_cf_sptr.**set_min_noutput_items**(*rms_cf_sptr self*, *int m*)

rms_cf_sptr.**set_thread_priority**(*rms_cf_sptr self*, *int priority*) → int

rms_cf_sptr.**thread_priority**(*rms_cf_sptr self*) → int

gnuradio.blocks.**rms_ff**(*double alpha=0.0001*) → rms_ff_sptr

RMS average power.

Constructor Specific Documentation:

Make an RMS calc. block.

Parameters: **alpha** – gain for running average filter.

rms_ff_sptr.**active_thread_priority**(*rms_ff_sptr self*) → int

rms_ff_sptr.**declare_sample_delay**(*rms_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(rms_ff_sptr self, unsigned int delay)

rms_ff_sptr.**message_subscribers**(*rms_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

rms_ff_sptr.**min_noutput_items**(*rms_ff_sptr self*) → int

rms_ff_sptr.**pc_input_buffers_full_avg**(*rms_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(rms_ff_sptr self) -> pmt_vector_float

rms_ff_sptr.**pc_noutput_items_avg**(*rms_ff_sptr self*) → float

rms_ff_sptr.**pc_nproduced_avg**(*rms_ff_sptr self*) → float

rms_ff_sptr.**pc_output_buffers_full_avg**(*rms_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(rms_ff_sptr self) -> pmt_vector_float

rms_ff_sptr.**pc_throughput_avg**(*rms_ff_sptr self*) → float

rms_ff_sptr.**pc_work_time_avg**(*rms_ff_sptr self*) → float

rms_ff_sptr.**pc_work_time_total**(*rms_ff_sptr self*) → float

rms_ff_sptr.**sample_delay**(*rms_ff_sptr self*, *int which*) → unsigned int

rms_ff_sptr.**set_alpha**(*rms_ff_sptr self*, *double alpha*)

rms_ff_sptr.**set_min_noutput_items**(*rms_ff_sptr self*, *int m*)

rms_ff_sptr.**set_thread_priority**(*rms_ff_sptr self*, *int priority*) → int

rms_ff_sptr.**thread_priority**(*rms_ff_sptr self*) → int

gnuradio.blocks.**rotator_cc**(*double phase_inc=0.0*) → rotator_cc_sptr

Complex rotator.

Constructor Specific Documentation:

Make an complex rotator block.

Parameters: **phase_inc** – rotational velocity

rotator_cc_sptr.**active_thread_priority**(*rotator_cc_sptr self*) → int

rotator_cc_sptr.**declare_sample_delay**(*rotator_cc_sptr self*, *int which*, *int delay*)

declare_sample_delay(rotator_cc_sptr self, unsigned int delay)

rotator_cc_sptr.**message_subscribers**(*rotator_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

rotator_cc_sptr.**min_noutput_items**(*rotator_cc_sptr self*) → int

rotator_cc_sptr.**pc_input_buffers_full_avg**(*rotator_cc_sptr self*, *int which*) → float

pc_input_buffers_full_avg(rotator_cc_sptr self) -> pmt_vector_float

rotator_cc_sptr.**pc_noutput_items_avg**(*rotator_cc_sptr self*) → float

rotator_cc_sptr.**pc_nproduced_avg**(*rotator_cc_sptr self*) → float

rotator_cc_sptr.**pc_output_buffers_full_avg**(*rotator_cc_sptr self*, *int which*) → float

pc_output_buffers_full_avg(rotator_cc_sptr self) -> pmt_vector_float

rotator_cc_sptr.**pc_throughput_avg**(*rotator_cc_sptr self*) → float

rotator_cc_sptr.**pc_work_time_avg**(*rotator_cc_sptr self*) → float

rotator_cc_sptr.**pc_work_time_total**(*rotator_cc_sptr self*) → float

rotator_cc_sptr.**sample_delay**(*rotator_cc_sptr self*, *int which*) → unsigned int

rotator_cc_sptr.**set_min_noutput_items**(*rotator_cc_sptr self*, *int m*)

rotator_cc_sptr.**set_phase_inc**(*rotator_cc_sptr self*, *double phase_inc*)

rotator_cc_sptr.**set_thread_priority**(*rotator_cc_sptr self*, *int priority*) → int

rotator_cc_sptr.**thread_priority**(*rotator_cc_sptr self*) → int

gnuradio.blocks.**sample_and_hold_bb**() → sample_and_hold_bb_sptr

sample and hold circuit

Samples the data stream (input stream 0) and holds the value if the control signal is 1 (intput stream 1).

Constructor Specific Documentation:

sample_and_hold_bb_sptr.**active_thread_priority**(*sample_and_hold_bb_sptr self*) → int

sample_and_hold_bb_sptr.**declare_sample_delay**(*sample_and_hold_bb_sptr self*, *int which*, *int delay*)

declare_sample_delay(sample_and_hold_bb_sptr self, unsigned int delay)

sample_and_hold_bb_sptr.**message_subscribers**(*sample_and_hold_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sample_and_hold_bb_sptr.**min_noutput_items**(*sample_and_hold_bb_sptr self*) → int

sample_and_hold_bb_sptr.**pc_input_buffers_full_avg**(*sample_and_hold_bb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sample_and_hold_bb_sptr self) -> pmt_vector_float

sample_and_hold_bb_sptr.**pc_noutput_items_avg**(*sample_and_hold_bb_sptr self*) → float

sample_and_hold_bb_sptr.**pc_nproduced_avg**(*sample_and_hold_bb_sptr self*) → float

sample_and_hold_bb_sptr.**pc_output_buffers_full_avg**(*sample_and_hold_bb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sample_and_hold_bb_sptr self) -> pmt_vector_float

sample_and_hold_bb_sptr.**pc_throughput_avg**(*sample_and_hold_bb_sptr self*) → float

sample_and_hold_bb_sptr.**pc_work_time_avg**(*sample_and_hold_bb_sptr self*) → float

sample_and_hold_bb_sptr.**pc_work_time_total**(*sample_and_hold_bb_sptr self*) → float

sample_and_hold_bb_sptr.**sample_delay**(*sample_and_hold_bb_sptr self*, *int which*) → unsigned int

sample_and_hold_bb_sptr.**set_min_noutput_items**(*sample_and_hold_bb_sptr self*, *int m*)

sample_and_hold_bb_sptr.**set_thread_priority**(*sample_and_hold_bb_sptr self*, *int priority*) → int

sample_and_hold_bb_sptr.**thread_priority**(*sample_and_hold_bb_sptr self*) → int

gnuradio.blocks.**sample_and_hold_ff**() → sample_and_hold_ff_sptr

sample and hold circuit

Samples the data stream (input stream 0) and holds the value if the control signal is 1 (intput stream 1).

Constructor Specific Documentation:

sample_and_hold_ff_sptr.**active_thread_priority**(*sample_and_hold_ff_sptr self*) → int

sample_and_hold_ff_sptr.**declare_sample_delay**(*sample_and_hold_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(sample_and_hold_ff_sptr self, unsigned int delay)

sample_and_hold_ff_sptr.**message_subscribers**(*sample_and_hold_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sample_and_hold_ff_sptr.**min_noutput_items**(*sample_and_hold_ff_sptr self*) → int

sample_and_hold_ff_sptr.**pc_input_buffers_full_avg**(*sample_and_hold_ff_sptr self*, *int which*) →

float

pc_input_buffers_full_avg(sample_and_hold_ff_sptr self) -> pmt_vector_float

sample_and_hold_ff_sptr.**pc_noutput_items_avg**(*sample_and_hold_ff_sptr self*) → float

sample_and_hold_ff_sptr.**pc_nproduced_avg**(*sample_and_hold_ff_sptr self*) → float

sample_and_hold_ff_sptr.**pc_output_buffers_full_avg**(*sample_and_hold_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sample_and_hold_ff_sptr self) -> pmt_vector_float

sample_and_hold_ff_sptr.**pc_throughput_avg**(*sample_and_hold_ff_sptr self*) → float

sample_and_hold_ff_sptr.**pc_work_time_avg**(*sample_and_hold_ff_sptr self*) → float

sample_and_hold_ff_sptr.**pc_work_time_total**(*sample_and_hold_ff_sptr self*) → float

sample_and_hold_ff_sptr.**sample_delay**(*sample_and_hold_ff_sptr self*, *int which*) → unsigned int

sample_and_hold_ff_sptr.**set_min_noutput_items**(*sample_and_hold_ff_sptr self*, *int m*)

sample_and_hold_ff_sptr.**set_thread_priority**(*sample_and_hold_ff_sptr self*, *int priority*) → int

sample_and_hold_ff_sptr.**thread_priority**(*sample_and_hold_ff_sptr self*) → int

gnuradio.blocks.**sample_and_hold_ii**() → sample_and_hold_ii_sptr

sample and hold circuit

Samples the data stream (input stream 0) and holds the value if the control signal is 1 (intput stream 1).

Constructor Specific Documentation:

sample_and_hold_ii_sptr.**active_thread_priority**(*sample_and_hold_ii_sptr self*) → int

sample_and_hold_ii_sptr.**declare_sample_delay**(*sample_and_hold_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(sample_and_hold_ii_sptr self, unsigned int delay)

sample_and_hold_ii_sptr.**message_subscribers**(*sample_and_hold_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sample_and_hold_ii_sptr.**min_noutput_items**(*sample_and_hold_ii_sptr self*) → int

sample_and_hold_ii_sptr.**pc_input_buffers_full_avg**(*sample_and_hold_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sample_and_hold_ii_sptr self) -> pmt_vector_float

sample_and_hold_ii_sptr.**pc_noutput_items_avg**(*sample_and_hold_ii_sptr self*) → float

sample_and_hold_ii_sptr.**pc_nproduced_avg**(*sample_and_hold_ii_sptr self*) → float

sample_and_hold_ii_sptr.**pc_output_buffers_full_avg**(*sample_and_hold_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sample_and_hold_ii_sptr self) -> pmt_vector_float

sample_and_hold_ii_sptr.**pc_throughput_avg**(*sample_and_hold_ii_sptr self*) → float

sample_and_hold_ii_sptr.**pc_work_time_avg**(*sample_and_hold_ii_sptr self*) → float

sample_and_hold_ii_sptr.**pc_work_time_total**(*sample_and_hold_ii_sptr self*) → float

sample_and_hold_ii_sptr.**sample_delay**(*sample_and_hold_ii_sptr self*, *int which*) → unsigned int

sample_and_hold_ii_sptr.**set_min_noutput_items**(*sample_and_hold_ii_sptr self*, *int m*)

sample_and_hold_ii_sptr.**set_thread_priority**(*sample_and_hold_ii_sptr self*, *int priority*) → int

sample_and_hold_ii_sptr.**thread_priority**(*sample_and_hold_ii_sptr self*) → int

gnuradio.blocks.**sample_and_hold_ss**() → sample_and_hold_ss_sptr

sample and hold circuit

Samples the data stream (input stream 0) and holds the value if the control signal is 1 (intput stream 1).

Constructor Specific Documentation:

sample_and_hold_ss_sptr.**active_thread_priority**(*sample_and_hold_ss_sptr self*) → int

sample_and_hold_ss_sptr.**declare_sample_delay**(*sample_and_hold_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(sample_and_hold_ss_sptr self, unsigned int delay)

sample_and_hold_ss_sptr.**message_subscribers**(*sample_and_hold_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sample_and_hold_ss_sptr.**min_noutput_items**(*sample_and_hold_ss_sptr self*) → int

sample_and_hold_ss_sptr.**pc_input_buffers_full_avg**(*sample_and_hold_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sample_and_hold_ss_sptr self) -> pmt_vector_float

sample_and_hold_ss_sptr.**pc_noutput_items_avg**(*sample_and_hold_ss_sptr self*) → float

sample_and_hold_ss_sptr.**pc_nproduced_avg**(*sample_and_hold_ss_sptr self*) → float

sample_and_hold_ss_sptr.**pc_output_buffers_full_avg**(*sample_and_hold_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sample_and_hold_ss_sptr self) -> pmt_vector_float

sample_and_hold_ss_sptr.**pc_throughput_avg**(*sample_and_hold_ss_sptr self*) → float

sample_and_hold_ss_sptr.**pc_work_time_avg**(*sample_and_hold_ss_sptr self*) → float

sample_and_hold_ss_sptr.**pc_work_time_total**(*sample_and_hold_ss_sptr self*) → float

sample_and_hold_ss_sptr.**sample_delay**(*sample_and_hold_ss_sptr self*, *int which*) → unsigned int

sample_and_hold_ss_sptr.**set_min_noutput_items**(*sample_and_hold_ss_sptr self*, *int m*)

sample_and_hold_ss_sptr.**set_thread_priority**(*sample_and_hold_ss_sptr self*, *int priority*) → int

sample_and_hold_ss_sptr.**thread_priority**(*sample_and_hold_ss_sptr self*) → int

gnuradio.blocks.**short_to_char**(*size_t vlen=1*) → short_to_char_sptr

Convert stream of shorts to a stream of chars.

This block strips off the least significant byte from the short value.

[0x00ff, 0x0ff0, 0xff00] => [0x00, 0x0f, 0xff]

Converts length vectors of input short samples to chars, dividing each element by 256.

Constructor Specific Documentation:

Build a short to char block.

Parameters: **vlen** – vector length of data streams.

short_to_char_sptr.**active_thread_priority**(*short_to_char_sptr self*) → int

short_to_char_sptr.**declare_sample_delay**(*short_to_char_sptr self*, *int which*, *int delay*)

declare_sample_delay(short_to_char_sptr self, unsigned int delay)

short_to_char_sptr.**message_subscribers**(*short_to_char_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

short_to_char_sptr.**min_noutput_items**(*short_to_char_sptr self*) → int

short_to_char_sptr.**pc_input_buffers_full_avg**(*short_to_char_sptr self*, *int which*) → float

pc_input_buffers_full_avg(short_to_char_sptr self) -> pmt_vector_float

short_to_char_sptr.**pc_noutput_items_avg**(*short_to_char_sptr self*) → float

short_to_char_sptr.**pc_nproduced_avg**(*short_to_char_sptr self*) → float

short_to_char_sptr.**pc_output_buffers_full_avg**(*short_to_char_sptr self*, *int which*) → float

pc_output_buffers_full_avg(short_to_char_sptr self) -> pmt_vector_float

short_to_char_sptr.**pc_throughput_avg**(*short_to_char_sptr self*) → float

short_to_char_sptr.**pc_work_time_avg**(*short_to_char_sptr self*) → float

short_to_char_sptr.**pc_work_time_total**(*short_to_char_sptr self*) → float

short_to_char_sptr.**sample_delay**(*short_to_char_sptr self*, *int which*) → unsigned int

short_to_char_sptr.**set_min_noutput_items**(*short_to_char_sptr self*, *int m*)

short_to_char_sptr.**set_thread_priority**(*short_to_char_sptr self*, *int priority*) → int

short_to_char_sptr.**thread_priority**(*short_to_char_sptr self*) → int

gnuradio.blocks.**short_to_float**(*size_t vlen=1*, *float scale=1.0*) → short_to_float_sptr

Convert stream of shorts to a stream of floats.

Constructor Specific Documentation:

Build a short to float block.

Parameters:
- **vlen** – vector length of data streams.
- **scale** – a scalar divider to change the output signal scale.

short_to_float_sptr.**active_thread_priority**(*short_to_float_sptr self*) → int

short_to_float_sptr.**declare_sample_delay**(*short_to_float_sptr self*, *int which*, *int delay*)

declare_sample_delay(short_to_float_sptr self, unsigned int delay)

short_to_float_sptr.**message_subscribers**(*short_to_float_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

short_to_float_sptr.**min_noutput_items**(*short_to_float_sptr self*) → int

short_to_float_sptr.**pc_input_buffers_full_avg**(*short_to_float_sptr self*, *int which*) → float

pc_input_buffers_full_avg(short_to_float_sptr self) -> pmt_vector_float

short_to_float_sptr.**pc_noutput_items_avg**(*short_to_float_sptr self*) → float

short_to_float_sptr.**pc_nproduced_avg**(*short_to_float_sptr self*) → float

short_to_float_sptr.**pc_output_buffers_full_avg**(*short_to_float_sptr self*, *int which*) → float

pc_output_buffers_full_avg(short_to_float_sptr self) -> pmt_vector_float

short_to_float_sptr.**pc_throughput_avg**(*short_to_float_sptr self*) → float

short_to_float_sptr.**pc_work_time_avg**(*short_to_float_sptr self*) → float

short_to_float_sptr.**pc_work_time_total**(*short_to_float_sptr self*) → float

short_to_float_sptr.**sample_delay**(*short_to_float_sptr self*, *int which*) → unsigned int

short_to_float_sptr.**scale**(*short_to_float_sptr self*) → float

Get the scalar divider value.

short_to_float_sptr.**set_min_noutput_items**(*short_to_float_sptr self*, *int m*)

short_to_float_sptr.**set_scale**(*short_to_float_sptr self*, *float scale*)

Set the scalar divider value.

short_to_float_sptr.**set_thread_priority**(*short_to_float_sptr self*, *int priority*) → int

short_to_float_sptr.**thread_priority**(*short_to_float_sptr self*) → int

gnuradio.blocks.**skiphead**(*size_t itemsize*, *uint64_t nitems_to_skip*) → skiphead_sptr

skips the first N items, from then on copies items to the output

Useful for building test cases and sources which have metadata or junk at the start

Constructor Specific Documentation:

Parameters:
- **itemsize** –
- **nitems_to_skip** –

skiphead_sptr.**active_thread_priority**(*skiphead_sptr self*) → int

skiphead_sptr.**declare_sample_delay**(*skiphead_sptr self*, *int which*, *int delay*)

declare_sample_delay(skiphead_sptr self, unsigned int delay)

skiphead_sptr.**message_subscribers**(*skiphead_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

skiphead_sptr.**min_noutput_items**(*skiphead_sptr self*) → int

skiphead_sptr.**pc_input_buffers_full_avg**(*skiphead_sptr self*, *int which*) → float

pc_input_buffers_full_avg(skiphead_sptr self) -> pmt_vector_float

skiphead_sptr.**pc_noutput_items_avg**(*skiphead_sptr self*) → float

skiphead_sptr.**pc_nproduced_avg**(*skiphead_sptr self*) → float

skiphead_sptr.**pc_output_buffers_full_avg**(*skiphead_sptr self*, *int which*) → float

pc_output_buffers_full_avg(skiphead_sptr self) -> pmt_vector_float

skiphead_sptr.**pc_throughput_avg**(*skiphead_sptr self*) → float

skiphead_sptr.**pc_work_time_avg**(*skiphead_sptr self*) → float

skiphead_sptr.**pc_work_time_total**(*skiphead_sptr self*) → float

skiphead_sptr.**sample_delay**(*skiphead_sptr self*, *int which*) → unsigned int

skiphead_sptr.**set_min_noutput_items**(*skiphead_sptr self*, *int m*)

skiphead_sptr.**set_thread_priority**(*skiphead_sptr self*, *int priority*) → int

skiphead_sptr.**thread_priority**(*skiphead_sptr self*) → int

gnuradio.blocks.**socket_pdu**(*std::string type*, *std::string addr*, *std::string port*, *int MTU=10000*, *bool tcp_no_delay=False*) → socket_pdu_sptr

Creates socket interface and translates traffic to PDUs.

Constructor Specific Documentation:

Construct a SOCKET PDU interface.

Parameters:
- **type** – “TCP_SERVER”, “TCP_CLIENT”, “UDP_SERVER”, or “UDP_CLIENT”
- **addr** – network address to use
- **port** – network port to use
- **MTU** – maximum transmission unit
- **tcp_no_delay** – TCP No Delay option (set to True to disable Nagle algorithm)

socket_pdu_sptr.**active_thread_priority**(*socket_pdu_sptr self*) → int

socket_pdu_sptr.**declare_sample_delay**(*socket_pdu_sptr self*, *int which*, *int delay*)

declare_sample_delay(socket_pdu_sptr self, unsigned int delay)

socket_pdu_sptr.**message_subscribers**(*socket_pdu_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

socket_pdu_sptr.**min_noutput_items**(*socket_pdu_sptr self*) → int

socket_pdu_sptr.**pc_input_buffers_full_avg**(*socket_pdu_sptr self*, *int which*) → float

pc_input_buffers_full_avg(socket_pdu_sptr self) -> pmt_vector_float

socket_pdu_sptr.**pc_noutput_items_avg**(*socket_pdu_sptr self*) → float

socket_pdu_sptr.**pc_nproduced_avg**(*socket_pdu_sptr self*) → float

socket_pdu_sptr.**pc_output_buffers_full_avg**(*socket_pdu_sptr self*, *int which*) → float

pc_output_buffers_full_avg(socket_pdu_sptr self) -> pmt_vector_float

socket_pdu_sptr.**pc_throughput_avg**(*socket_pdu_sptr self*) → float

socket_pdu_sptr.**pc_work_time_avg**(*socket_pdu_sptr self*) → float

socket_pdu_sptr.**pc_work_time_total**(*socket_pdu_sptr self*) → float

socket_pdu_sptr.**sample_delay**(*socket_pdu_sptr self*, *int which*) → unsigned int

socket_pdu_sptr.**set_min_noutput_items**(*socket_pdu_sptr self*, *int m*)

socket_pdu_sptr.**set_thread_priority**(*socket_pdu_sptr self*, *int priority*) → int

socket_pdu_sptr.**thread_priority**(*socket_pdu_sptr self*) → int

gnuradio.blocks.**stream_mux**(*size_t itemsize*, *std::vector< int, std::allocator< int > > const & lengths*) → stream_mux_sptr

Stream muxing block to multiplex many streams into one with a specified format.

Muxes N streams together producing an output stream that contains N0 items from the first stream, N1 items from the second, etc. and repeats:

[N0, N1, N2, ..., Nm, N0, N1, ...]

Constructor Specific Documentation:

Creates a stream muxing block to multiplex many streams into one with a specified format.

Parameters:
- **itemsize** – the item size of the stream
- **lengths** – a vector (list/tuple) specifying the number of items from each stream the mux together. Warning: this requires that at least as many items per stream are available or the system will wait indefinitely for the items.

stream_mux_sptr.**active_thread_priority**(*stream_mux_sptr self*) → int

stream_mux_sptr.**declare_sample_delay**(*stream_mux_sptr self*, *int which*, *int delay*)
declare_sample_delay(stream_mux_sptr self, unsigned int delay)

stream_mux_sptr.**message_subscribers**(*stream_mux_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

stream_mux_sptr.**min_noutput_items**(*stream_mux_sptr self*) → int

stream_mux_sptr.**pc_input_buffers_full_avg**(*stream_mux_sptr self*, *int which*) → float
pc_input_buffers_full_avg(stream_mux_sptr self) -> pmt_vector_float

stream_mux_sptr.**pc_noutput_items_avg**(*stream_mux_sptr self*) → float

stream_mux_sptr.**pc_nproduced_avg**(*stream_mux_sptr self*) → float

stream_mux_sptr.**pc_output_buffers_full_avg**(*stream_mux_sptr self*, *int which*) → float
pc_output_buffers_full_avg(stream_mux_sptr self) -> pmt_vector_float

stream_mux_sptr.**pc_throughput_avg**(*stream_mux_sptr self*) → float

stream_mux_sptr.**pc_work_time_avg**(*stream_mux_sptr self*) → float

stream_mux_sptr.**pc_work_time_total**(*stream_mux_sptr self*) → float

stream_mux_sptr.**sample_delay**(*stream_mux_sptr self*, *int which*) → unsigned int

stream_mux_sptr.**set_min_noutput_items**(*stream_mux_sptr self*, *int m*)

stream_mux_sptr.**set_thread_priority**(*stream_mux_sptr self*, *int priority*) → int

stream_mux_sptr.**thread_priority**(*stream_mux_sptr self*) → int

gnuradio.blocks.**stream_to_streams**(*size_t itemsize*, *size_t nstreams*) → stream_to_streams_sptr
convert a stream of items into a N streams of items

Converts a stream of N items into N streams of 1 item. Repeat ad infinitum.

Constructor Specific Documentation:

Make a stream-to-streams block.

Parameters:
- **itemsize** – the item size of the stream
- **nstreams** – number of streams to split input into

stream_to_streams_sptr.**active_thread_priority**(*stream_to_streams_sptr self*) → int

stream_to_streams_sptr.**declare_sample_delay**(*stream_to_streams_sptr self*, *int which*, *int delay*)
declare_sample_delay(stream_to_streams_sptr self, unsigned int delay)

stream_to_streams_sptr.**message_subscribers**(*stream_to_streams_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

stream_to_streams_sptr.**min_noutput_items**(*stream_to_streams_sptr self*) → int

stream_to_streams_sptr.**pc_input_buffers_full_avg**(*stream_to_streams_sptr self*, *int which*) → float
pc_input_buffers_full_avg(stream_to_streams_sptr self) -> pmt_vector_float

stream_to_streams_sptr.**pc_noutput_items_avg**(*stream_to_streams_sptr self*) → float

stream_to_streams_sptr.**pc_nproduced_avg**(*stream_to_streams_sptr self*) → float

stream_to_streams_sptr.**pc_output_buffers_full_avg**(*stream_to_streams_sptr self*, *int which*) → float
pc_output_buffers_full_avg(stream_to_streams_sptr self) -> pmt_vector_float

stream_to_streams_sptr.**pc_throughput_avg**(*stream_to_streams_sptr self*) → float

stream_to_streams_sptr.**pc_work_time_avg**(*stream_to_streams_sptr self*) → float

stream_to_streams_sptr.**pc_work_time_total**(*stream_to_streams_sptr self*) → float

stream_to_streams_sptr.**sample_delay**(*stream_to_streams_sptr self*, *int which*) → unsigned int

stream_to_streams_sptr.**set_min_noutput_items**(*stream_to_streams_sptr self*, *int m*)

stream_to_streams_sptr.**set_thread_priority**(*stream_to_streams_sptr self*, *int priority*) → int

stream_to_streams_sptr.**thread_priority**(*stream_to_streams_sptr self*) → int

gnuradio.blocks.**stream_to_tagged_stream**(*size_t itemsize*, *int vlen*, *unsigned int packet_len*, *std::string const & len_tag_key*) → stream_to_tagged_stream_sptr

Converts a regular stream into a tagged stream.

All this block does is add length tags in regular intervals. It can be used to connect a regular stream to a gr::tagged_stream_block.

This block is meant to be connected directly to a tagged stream block. If there are blocks between this block and a tagged stream block, make sure they either don't change the rate, or modify the tag value to make sure the length tags actually represent the packet length.

Constructor Specific Documentation:

Parameters:
- **itemsize** – Item size
- **vlen** – Vector length of the input items. Note that one vector is one item.
- **packet_len** – Number of items per tagged stream packet. One tag is written every items.
- **len_tag_key** – Key of the length tag.

stream_to_tagged_stream_sptr.**active_thread_priority**(*stream_to_tagged_stream_sptr self*) → int

stream_to_tagged_stream_sptr.**declare_sample_delay**(*stream_to_tagged_stream_sptr self*, *int which*, *int delay*)

declare_sample_delay(stream_to_tagged_stream_sptr self, unsigned int delay)

stream_to_tagged_stream_sptr.**message_subscribers**(*stream_to_tagged_stream_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

stream_to_tagged_stream_sptr.**min_noutput_items**(*stream_to_tagged_stream_sptr self*) → int

stream_to_tagged_stream_sptr.**pc_input_buffers_full_avg**(*stream_to_tagged_stream_sptr self*, *int which*) → float

pc_input_buffers_full_avg(stream_to_tagged_stream_sptr self) -> pmt_vector_float

stream_to_tagged_stream_sptr.**pc_noutput_items_avg**(*stream_to_tagged_stream_sptr self*) → float

stream_to_tagged_stream_sptr.**pc_nproduced_avg**(*stream_to_tagged_stream_sptr self*) → float

stream_to_tagged_stream_sptr.**pc_output_buffers_full_avg**(*stream_to_tagged_stream_sptr self*, *int which*) → float

pc_output_buffers_full_avg(stream_to_tagged_stream_sptr self) -> pmt_vector_float

stream_to_tagged_stream_sptr.**pc_throughput_avg**(*stream_to_tagged_stream_sptr self*) → float

stream_to_tagged_stream_sptr.**pc_work_time_avg**(*stream_to_tagged_stream_sptr self*) → float

stream_to_tagged_stream_sptr.**pc_work_time_total**(*stream_to_tagged_stream_sptr self*) → float

stream_to_tagged_stream_sptr.**sample_delay**(*stream_to_tagged_stream_sptr self*, *int which*) → unsigned int

stream_to_tagged_stream_sptr.**set_min_noutput_items**(*stream_to_tagged_stream_sptr self*, *int m*)

stream_to_tagged_stream_sptr.**set_packet_len**(*stream_to_tagged_stream_sptr self*, *unsigned int packet_len*)

stream_to_tagged_stream_sptr.**set_packet_len_pmt**(*stream_to_tagged_stream_sptr self*, *unsigned int packet_len*)

stream_to_tagged_stream_sptr.**set_thread_priority**(*stream_to_tagged_stream_sptr self*, *int priority*) → int

stream_to_tagged_stream_sptr.**thread_priority**(*stream_to_tagged_stream_sptr self*) → int

gnuradio.blocks.**stream_to_vector**(*size_t itemsize*, *size_t nitems_per_block*) → stream_to_vector_sptr

convert a stream of items into a stream of gnuradio/blocks containing nitems_per_block

Constructor Specific Documentation:

Make a stream-to-vector block.

Parameters:
- **itemsize** – the item size of the stream
- **nitems_per_block** – number of items to put into each vector (vector size)

stream_to_vector_sptr.**active_thread_priority**(*stream_to_vector_sptr self*) → int

stream_to_vector_sptr.**declare_sample_delay**(*stream_to_vector_sptr self*, *int which*, *int delay*)
declare_sample_delay(stream_to_vector_sptr self, unsigned int delay)

stream_to_vector_sptr.**message_subscribers**(*stream_to_vector_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

stream_to_vector_sptr.**min_noutput_items**(*stream_to_vector_sptr self*) → int

stream_to_vector_sptr.**pc_input_buffers_full_avg**(*stream_to_vector_sptr self*, *int which*) → float
pc_input_buffers_full_avg(stream_to_vector_sptr self) -> pmt_vector_float

stream_to_vector_sptr.**pc_noutput_items_avg**(*stream_to_vector_sptr self*) → float

stream_to_vector_sptr.**pc_nproduced_avg**(*stream_to_vector_sptr self*) → float

stream_to_vector_sptr.**pc_output_buffers_full_avg**(*stream_to_vector_sptr self*, *int which*) → float
pc_output_buffers_full_avg(stream_to_vector_sptr self) -> pmt_vector_float

stream_to_vector_sptr.**pc_throughput_avg**(*stream_to_vector_sptr self*) → float

stream_to_vector_sptr.**pc_work_time_avg**(*stream_to_vector_sptr self*) → float

stream_to_vector_sptr.**pc_work_time_total**(*stream_to_vector_sptr self*) → float

stream_to_vector_sptr.**sample_delay**(*stream_to_vector_sptr self*, *int which*) → unsigned int

stream_to_vector_sptr.**set_min_noutput_items**(*stream_to_vector_sptr self*, *int m*)

stream_to_vector_sptr.**set_thread_priority**(*stream_to_vector_sptr self*, *int priority*) → int

stream_to_vector_sptr.**thread_priority**(*stream_to_vector_sptr self*) → int

gnuradio.blocks.**streams_to_stream**(*size_t itemsize*, *size_t nstreams*) → streams_to_stream_sptr
Convert N streams of 1 item into a 1 stream of N items.

Convert N streams of 1 item into 1 stream of N items. Repeat ad infinitum.

Constructor Specific Documentation:

Make a streams-to-stream block.

Parameters:
- **itemsize** – the item size of the stream
- **nstreams** – number of streams to combine

streams_to_stream_sptr.**active_thread_priority**(*streams_to_stream_sptr self*) → int

streams_to_stream_sptr.**declare_sample_delay**(*streams_to_stream_sptr self*, *int which*, *int delay*)
declare_sample_delay(streams_to_stream_sptr self, unsigned int delay)

streams_to_stream_sptr.**message_subscribers**(*streams_to_stream_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

streams_to_stream_sptr.**min_noutput_items**(*streams_to_stream_sptr self*) → int

streams_to_stream_sptr.**pc_input_buffers_full_avg**(*streams_to_stream_sptr self*, *int which*) → float
pc_input_buffers_full_avg(streams_to_stream_sptr self) -> pmt_vector_float

streams_to_stream_sptr.**pc_noutput_items_avg**(*streams_to_stream_sptr self*) → float

streams_to_stream_sptr.**pc_nproduced_avg**(*streams_to_stream_sptr self*) → float

streams_to_stream_sptr.**pc_output_buffers_full_avg**(*streams_to_stream_sptr self*, *int which*) → float
pc_output_buffers_full_avg(streams_to_stream_sptr self) -> pmt_vector_float

streams_to_stream_sptr.**pc_throughput_avg**(*streams_to_stream_sptr self*) → float

streams_to_stream_sptr.**pc_work_time_avg**(*streams_to_stream_sptr self*) → float

streams_to_stream_sptr.**pc_work_time_total**(*streams_to_stream_sptr self*) → float

streams_to_stream_sptr.**sample_delay**(*streams_to_stream_sptr self*, *int which*) → unsigned int

streams_to_stream_sptr.**set_min_noutput_items**(*streams_to_stream_sptr self*, *int m*)

streams_to_stream_sptr.**set_thread_priority**(*streams_to_stream_sptr self*, *int priority*) → int

streams_to_stream_sptr.**thread_priority**(*streams_to_stream_sptr self*) → int

gnuradio.blocks.**streams_to_vector**(*size_t itemsize*, *size_t nstreams*) → streams_to_vector_sptr

convert N streams of items to 1 stream of vector length N

Constructor Specific Documentation:

Make a stream-to-vector block.

Parameters:
- **itemsize** – the item size of the stream
- **nstreams** – number of streams to combine into a vector (vector size)

streams_to_vector_sptr.**active_thread_priority**(*streams_to_vector_sptr self*) → int

streams_to_vector_sptr.**declare_sample_delay**(*streams_to_vector_sptr self*, *int which*, *int delay*)

declare_sample_delay(streams_to_vector_sptr self, unsigned int delay)

streams_to_vector_sptr.**message_subscribers**(*streams_to_vector_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

streams_to_vector_sptr.**min_noutput_items**(*streams_to_vector_sptr self*) → int

streams_to_vector_sptr.**pc_input_buffers_full_avg**(*streams_to_vector_sptr self*, *int which*) → float

pc_input_buffers_full_avg(streams_to_vector_sptr self) -> pmt_vector_float

streams_to_vector_sptr.**pc_noutput_items_avg**(*streams_to_vector_sptr self*) → float

streams_to_vector_sptr.**pc_nproduced_avg**(*streams_to_vector_sptr self*) → float

streams_to_vector_sptr.**pc_output_buffers_full_avg**(*streams_to_vector_sptr self*, *int which*) → float

pc_output_buffers_full_avg(streams_to_vector_sptr self) -> pmt_vector_float

streams_to_vector_sptr.**pc_throughput_avg**(*streams_to_vector_sptr self*) → float

streams_to_vector_sptr.**pc_work_time_avg**(*streams_to_vector_sptr self*) → float

streams_to_vector_sptr.**pc_work_time_total**(*streams_to_vector_sptr self*) → float

streams_to_vector_sptr.**sample_delay**(*streams_to_vector_sptr self*, *int which*) → unsigned int

streams_to_vector_sptr.**set_min_noutput_items**(*streams_to_vector_sptr self*, *int m*)

streams_to_vector_sptr.**set_thread_priority**(*streams_to_vector_sptr self*, *int priority*) → int

streams_to_vector_sptr.**thread_priority**(*streams_to_vector_sptr self*) → int

gnuradio.blocks.**stretch_ff**(*float lo*, *size_t vlen=1*) → stretch_ff_sptr

adjust y-range of an input vector by mapping to range (max-of-input, stipulated-min). Primarily for spectral signature matching by normalizing spectrum dynamic ranges.

Constructor Specific Documentation:

Make a stretch block.

Parameters:
- **lo** – Set low value for range.
- **vlen** – vector length of input stream.

stretch_ff_sptr.**active_thread_priority**(*stretch_ff_sptr self*) → int

stretch_ff_sptr.**declare_sample_delay**(*stretch_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(stretch_ff_sptr self, unsigned int delay)

stretch_ff_sptr.**lo**(*stretch_ff_sptr self*) → float

stretch_ff_sptr.**message_subscribers**(*stretch_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

stretch_ff_sptr.**min_noutput_items**(*stretch_ff_sptr self*) → int

stretch_ff_sptr.**pc_input_buffers_full_avg**(*stretch_ff_sptr self*, *int which*) → float
pc_input_buffers_full_avg(stretch_ff_sptr self) -> pmt_vector_float

stretch_ff_sptr.**pc_noutput_items_avg**(*stretch_ff_sptr self*) → float

stretch_ff_sptr.**pc_nproduced_avg**(*stretch_ff_sptr self*) → float

stretch_ff_sptr.**pc_output_buffers_full_avg**(*stretch_ff_sptr self*, *int which*) → float
pc_output_buffers_full_avg(stretch_ff_sptr self) -> pmt_vector_float

stretch_ff_sptr.**pc_throughput_avg**(*stretch_ff_sptr self*) → float

stretch_ff_sptr.**pc_work_time_avg**(*stretch_ff_sptr self*) → float

stretch_ff_sptr.**pc_work_time_total**(*stretch_ff_sptr self*) → float

stretch_ff_sptr.**sample_delay**(*stretch_ff_sptr self*, *int which*) → unsigned int

stretch_ff_sptr.**set_lo**(*stretch_ff_sptr self*, *float lo*)

stretch_ff_sptr.**set_min_noutput_items**(*stretch_ff_sptr self*, *int m*)

stretch_ff_sptr.**set_thread_priority**(*stretch_ff_sptr self*, *int priority*) → int

stretch_ff_sptr.**thread_priority**(*stretch_ff_sptr self*) → int

stretch_ff_sptr.**vlen**(*stretch_ff_sptr self*) → size_t

gnuradio.blocks.**sub_cc**(*size_t vlen=1*) → sub_cc_sptr
output = input_0 - input_1 - ...)

Subtract across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

sub_cc_sptr.**active_thread_priority**(*sub_cc_sptr self*) → int

sub_cc_sptr.**declare_sample_delay**(*sub_cc_sptr self*, *int which*, *int delay*)
declare_sample_delay(sub_cc_sptr self, unsigned int delay)

sub_cc_sptr.**message_subscribers**(*sub_cc_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sub_cc_sptr.**min_noutput_items**(*sub_cc_sptr self*) → int

sub_cc_sptr.**pc_input_buffers_full_avg**(*sub_cc_sptr self*, *int which*) → float
pc_input_buffers_full_avg(sub_cc_sptr self) -> pmt_vector_float

sub_cc_sptr.**pc_noutput_items_avg**(*sub_cc_sptr self*) → float

sub_cc_sptr.**pc_nproduced_avg**(*sub_cc_sptr self*) → float

sub_cc_sptr.**pc_output_buffers_full_avg**(*sub_cc_sptr self*, *int which*) → float
pc_output_buffers_full_avg(sub_cc_sptr self) -> pmt_vector_float

sub_cc_sptr.**pc_throughput_avg**(*sub_cc_sptr self*) → float

sub_cc_sptr.**pc_work_time_avg**(*sub_cc_sptr self*) → float

sub_cc_sptr.**pc_work_time_total**(*sub_cc_sptr self*) → float

sub_cc_sptr.**sample_delay**(*sub_cc_sptr self*, *int which*) → unsigned int

sub_cc_sptr.**set_min_noutput_items**(*sub_cc_sptr self*, *int m*)

sub_cc_sptr.**set_thread_priority**(*sub_cc_sptr self*, *int priority*) → int

sub_cc_sptr.**thread_priority**(*sub_cc_sptr self*) → int

gnuradio.blocks.**sub_ff**(*size_t vlen=1*) → sub_ff_sptr
output = input_0 - input_1 - ...)

Subtract across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

sub_ff_sptr.**active_thread_priority**(*sub_ff_sptr self*) → int

sub_ff_sptr.**declare_sample_delay**(*sub_ff_sptr self*, *int which*, *int delay*)
declare_sample_delay(sub_ff_sptr self, unsigned int delay)

sub_ff_sptr.**message_subscribers**(*sub_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sub_ff_sptr.**min_noutput_items**(*sub_ff_sptr self*) → int

sub_ff_sptr.**pc_input_buffers_full_avg**(*sub_ff_sptr self*, *int which*) → float
pc_input_buffers_full_avg(sub_ff_sptr self) -> pmt_vector_float

sub_ff_sptr.**pc_noutput_items_avg**(*sub_ff_sptr self*) → float

sub_ff_sptr.**pc_nproduced_avg**(*sub_ff_sptr self*) → float

sub_ff_sptr.**pc_output_buffers_full_avg**(*sub_ff_sptr self*, *int which*) → float
pc_output_buffers_full_avg(sub_ff_sptr self) -> pmt_vector_float

sub_ff_sptr.**pc_throughput_avg**(*sub_ff_sptr self*) → float

sub_ff_sptr.**pc_work_time_avg**(*sub_ff_sptr self*) → float

sub_ff_sptr.**pc_work_time_total**(*sub_ff_sptr self*) → float

sub_ff_sptr.**sample_delay**(*sub_ff_sptr self*, *int which*) → unsigned int

sub_ff_sptr.**set_min_noutput_items**(*sub_ff_sptr self*, *int m*)

sub_ff_sptr.**set_thread_priority**(*sub_ff_sptr self*, *int priority*) → int

sub_ff_sptr.**thread_priority**(*sub_ff_sptr self*) → int

gnuradio.blocks.**sub_ii**(*size_t vlen=1*) → sub_ii_sptr
output = input_0 - input_1 - ...)

Subtract across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

sub_ii_sptr.**active_thread_priority**(*sub_ii_sptr self*) → int

sub_ii_sptr.**declare_sample_delay**(*sub_ii_sptr self*, *int which*, *int delay*)
declare_sample_delay(sub_ii_sptr self, unsigned int delay)

sub_ii_sptr.**message_subscribers**(*sub_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sub_ii_sptr.**min_noutput_items**(*sub_ii_sptr self*) → int

sub_ii_sptr.**pc_input_buffers_full_avg**(*sub_ii_sptr self*, *int which*) → float
pc_input_buffers_full_avg(sub_ii_sptr self) -> pmt_vector_float

sub_ii_sptr.**pc_noutput_items_avg**(*sub_ii_sptr self*) → float

sub_ii_sptr.**pc_nproduced_avg**(*sub_ii_sptr self*) → float

sub_ii_sptr.**pc_output_buffers_full_avg**(*sub_ii_sptr self*, *int which*) → float
pc_output_buffers_full_avg(sub_ii_sptr self) -> pmt_vector_float

sub_ii_sptr.**pc_throughput_avg**(*sub_ii_sptr self*) → float

sub_ii_sptr.**pc_work_time_avg**(*sub_ii_sptr self*) → float

sub_ii_sptr.**pc_work_time_total**(*sub_ii_sptr self*) → float

sub_ii_sptr.**sample_delay**(*sub_ii_sptr self*, *int which*) → unsigned int

sub_ii_sptr.**set_min_noutput_items**(*sub_ii_sptr self*, *int m*)

sub_ii_sptr.**set_thread_priority**(*sub_ii_sptr self*, *int priority*) → int

sub_ii_sptr.**thread_priority**(*sub_ii_sptr self*) → int

gnuradio.blocks.**sub_ss**(*size_t vlen=1*) → sub_ss_sptr

output = input_0 - input_1 - ...)

Subtract across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

sub_ss_sptr.**active_thread_priority**(*sub_ss_sptr self*) → int

sub_ss_sptr.**declare_sample_delay**(*sub_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(sub_ss_sptr self, unsigned int delay)

sub_ss_sptr.**message_subscribers**(*sub_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

sub_ss_sptr.**min_noutput_items**(*sub_ss_sptr self*) → int

sub_ss_sptr.**pc_input_buffers_full_avg**(*sub_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(sub_ss_sptr self) -> pmt_vector_float

sub_ss_sptr.**pc_noutput_items_avg**(*sub_ss_sptr self*) → float

sub_ss_sptr.**pc_nproduced_avg**(*sub_ss_sptr self*) → float

sub_ss_sptr.**pc_output_buffers_full_avg**(*sub_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(sub_ss_sptr self) -> pmt_vector_float

sub_ss_sptr.**pc_throughput_avg**(*sub_ss_sptr self*) → float

sub_ss_sptr.**pc_work_time_avg**(*sub_ss_sptr self*) → float

sub_ss_sptr.**pc_work_time_total**(*sub_ss_sptr self*) → float

sub_ss_sptr.**sample_delay**(*sub_ss_sptr self*, *int which*) → unsigned int

sub_ss_sptr.**set_min_noutput_items**(*sub_ss_sptr self*, *int m*)

sub_ss_sptr.**set_thread_priority**(*sub_ss_sptr self*, *int priority*) → int

sub_ss_sptr.**thread_priority**(*sub_ss_sptr self*) → int

gnuradio.blocks.**tag_debug**(*size_t sizeof_stream_item*, *std::string const & name*, *std::string const & key_filter*) → tag_debug_sptr

Bit bucket that prints out any tag received.

This block collects all tags sent to it on all input ports and displays them to stdout in a formatted way. The parameter is used to identify which debug sink generated the tag, so when connecting a block to this debug sink, an appropriate name is something that identifies the input block.

This block otherwise acts as a NULL sink in that items from the input stream are ignored. It is designed to be able to attach to any block and watch all tags streaming out of that block for debugging purposes.

Specifying a key will allow this block to filter out all other tags and only display tags that match the given key. This can help clean up the output and allow you to focus in on a particular tag of interest.

The tags from the last call to this work function are stored and can be retrieved using the function ‘current_tags’.

Constructor Specific Documentation:

Build a tag debug block

Parameters:

- **sizeof_stream_item** – size of the items in the incoming stream.
- **name** – name to identify which debug sink generated the info.
- **key_filter** – Specify a tag’s key value to use as a filter.

tag_debug_sptr.**active_thread_priority**(*tag_debug_sptr self*) → int

tag_debug_sptr.**current_tags**(*tag_debug_sptr self*) → tags_vector_t

Returns a vector of tag_t items as of the last call to work.

tag_debug_sptr.**declare_sample_delay**(*tag_debug_sptr self*, *int which*, *int delay*)

declare_sample_delay(tag_debug_sptr self, unsigned int delay)

tag_debug_sptr.**key_filter**(*tag_debug_sptr self*) → std::string

Get the current filter key.

tag_debug_sptr.**message_subscribers**(*tag_debug_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tag_debug_sptr.**min_noutput_items**(*tag_debug_sptr self*) → int

tag_debug_sptr.**num_tags**(*tag_debug_sptr self*) → int

Return the total number of tags in the tag queue.

tag_debug_sptr.**pc_input_buffers_full_avg**(*tag_debug_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tag_debug_sptr self) -> pmt_vector_float

tag_debug_sptr.**pc_noutput_items_avg**(*tag_debug_sptr self*) → float

tag_debug_sptr.**pc_nproduced_avg**(*tag_debug_sptr self*) → float

tag_debug_sptr.**pc_output_buffers_full_avg**(*tag_debug_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tag_debug_sptr self) -> pmt_vector_float

tag_debug_sptr.**pc_throughput_avg**(*tag_debug_sptr self*) → float

tag_debug_sptr.**pc_work_time_avg**(*tag_debug_sptr self*) → float

tag_debug_sptr.**pc_work_time_total**(*tag_debug_sptr self*) → float

tag_debug_sptr.**sample_delay**(*tag_debug_sptr self*, *int which*) → unsigned int

tag_debug_sptr.**set_display**(*tag_debug_sptr self*, *bool d*)

Set the display of tags to stdout on/off.

tag_debug_sptr.**set_key_filter**(*tag_debug_sptr self*, *std::string const & key_filter*)

Set a new key to filter with.

tag_debug_sptr.**set_min_noutput_items**(*tag_debug_sptr self*, *int m*)

tag_debug_sptr.**set_thread_priority**(*tag_debug_sptr self*, *int priority*) → int

tag_debug_sptr.**thread_priority**(*tag_debug_sptr self*) → int

gnuradio.blocks.**tag_gate**(*size_t item_size*, *bool propagate_tags=False*) → tag_gate_sptr

Control tag propagation.

Use this block to stop tags from propagating.

Constructor Specific Documentation:

Parameters:
- **item_size** – Item size
- **propagate_tags** – Set this to true to allow tags to pass through this block.

tag_gate_sptr.**active_thread_priority**(*tag_gate_sptr self*) → int

tag_gate_sptr.**declare_sample_delay**(*tag_gate_sptr self*, *int which*, *int delay*)

declare_sample_delay(tag_gate_sptr self, unsigned int delay)

tag_gate_sptr.**message_subscribers**(*tag_gate_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tag_gate_sptr.**min_noutput_items**(*tag_gate_sptr self*) → int

tag_gate_sptr.**pc_input_buffers_full_avg**(*tag_gate_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tag_gate_sptr self) -> pmt_vector_float

tag_gate_sptr.**pc_noutput_items_avg**(*tag_gate_sptr self*) → float

tag_gate_sptr.**pc_nproduced_avg**(*tag_gate_sptr self*) → float

tag_gate_sptr.**pc_output_buffers_full_avg**(*tag_gate_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tag_gate_sptr self) -> pmt_vector_float

tag_gate_sptr.**pc_throughput_avg**(*tag_gate_sptr self*) → float

tag_gate_sptr.**pc_work_time_avg**(*tag_gate_sptr self*) → float

tag_gate_sptr.**pc_work_time_total**(*tag_gate_sptr self*) → float

tag_gate_sptr.**sample_delay**(*tag_gate_sptr self*, *int which*) → unsigned int

tag_gate_sptr.**set_min_noutput_items**(*tag_gate_sptr self*, *int m*)

tag_gate_sptr.**set_propagation**(*tag_gate_sptr self*, *bool propagate_tags*)

tag_gate_sptr.**set_single_key**(*tag_gate_sptr self*, *std::string const & single_key*)

Only gate stream tags with one specific key instead of all keys.

If set to “”, all tags will be affected by the gate. If set to “foo”, all tags with key different from “foo” will pass through.

tag_gate_sptr.**set_thread_priority**(*tag_gate_sptr self*, *int priority*) → int

tag_gate_sptr.**single_key**(*tag_gate_sptr self*) → std::string

Get the current single key.

tag_gate_sptr.**thread_priority**(*tag_gate_sptr self*) → int

gnuradio.blocks.**tagged_file_sink**(*size_t itemsize*, *double samp_rate*) → tagged_file_sink_sptr

A file sink that uses tags to save files.

The sink uses a tag with the key ‘burst’ to trigger the saving of the burst data to a new file. If the value of this tag is True, it will open a new file and start writing all incoming data to it. If the tag is False, it will close the file (if already opened). The file names are based on the time when the burst tag was seen. If there is an ‘rx_time’ tag (standard with UHD sources), that is used as the time. If no ‘rx_time’ tag is found, the new time is calculated based off the sample rate of the block.

Constructor Specific Documentation:

Build a tagged_file_sink block.

Parameters:
- **itemsize** – The item size of the input data stream.
- **samp_rate** – The sample rate used to determine the time difference between bursts

tagged_file_sink_sptr.**active_thread_priority**(*tagged_file_sink_sptr self*) → int

tagged_file_sink_sptr.**declare_sample_delay**(*tagged_file_sink_sptr self*, *int which*, *int delay*)

declare_sample_delay(tagged_file_sink_sptr self, unsigned int delay)

tagged_file_sink_sptr.**message_subscribers**(*tagged_file_sink_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tagged_file_sink_sptr.**min_noutput_items**(*tagged_file_sink_sptr self*) → int

tagged_file_sink_sptr.**pc_input_buffers_full_avg**(*tagged_file_sink_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tagged_file_sink_sptr self) -> pmt_vector_float

tagged_file_sink_sptr.**pc_noutput_items_avg**(*tagged_file_sink_sptr self*) → float

tagged_file_sink_sptr.**pc_nproduced_avg**(*tagged_file_sink_sptr self*) → float

tagged_file_sink_sptr.**pc_output_buffers_full_avg**(*tagged_file_sink_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tagged_file_sink_sptr self) -> pmt_vector_float

tagged_file_sink_sptr.**pc_throughput_avg**(*tagged_file_sink_sptr self*) → float

tagged_file_sink_sptr.**pc_work_time_avg**(*tagged_file_sink_sptr self*) → float

tagged_file_sink_sptr.**pc_work_time_total**(*tagged_file_sink_sptr self*) → float

tagged_file_sink_sptr.**sample_delay**(*tagged_file_sink_sptr self*, *int which*) → unsigned int

tagged_file_sink_sptr.**set_min_noutput_items**(*tagged_file_sink_sptr self*, *int m*)

tagged_file_sink_sptr.**set_thread_priority**(*tagged_file_sink_sptr self*, *int priority*) → int

tagged_file_sink_sptr.**thread_priority**(*tagged_file_sink_sptr self*) → int

gnuradio.blocks.**tagged_stream_align**(*size_t itemsize*, *std::string const & lengthtagname*) → tagged_stream_align_sptr

Align a stream to a tagged stream item.

Takes an unaligned stream of tagged stream items and aligns to the first item

Constructor Specific Documentation:

Make a tagged stream align

Parameters:
- **itemsize** – The size (in bytes) of the item datatype.
- **lengthtagname** – Name of the TSB’s length tag key.

tagged_stream_align_sptr.**active_thread_priority**(*tagged_stream_align_sptr self*) → int

tagged_stream_align_sptr.**declare_sample_delay**(*tagged_stream_align_sptr self*, *int which*, *int delay*)

declare_sample_delay(tagged_stream_align_sptr self, unsigned int delay)

tagged_stream_align_sptr.**message_subscribers**(*tagged_stream_align_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tagged_stream_align_sptr.**min_noutput_items**(*tagged_stream_align_sptr self*) → int

tagged_stream_align_sptr.**pc_input_buffers_full_avg**(*tagged_stream_align_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tagged_stream_align_sptr self) -> pmt_vector_float

tagged_stream_align_sptr.**pc_noutput_items_avg**(*tagged_stream_align_sptr self*) → float

tagged_stream_align_sptr.**pc_nproduced_avg**(*tagged_stream_align_sptr self*) → float

tagged_stream_align_sptr.**pc_output_buffers_full_avg**(*tagged_stream_align_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tagged_stream_align_sptr self) -> pmt_vector_float

tagged_stream_align_sptr.**pc_throughput_avg**(*tagged_stream_align_sptr self*) → float

tagged_stream_align_sptr.**pc_work_time_avg**(*tagged_stream_align_sptr self*) → float

tagged_stream_align_sptr.**pc_work_time_total**(*tagged_stream_align_sptr self*) → float

tagged_stream_align_sptr.**sample_delay**(*tagged_stream_align_sptr self*, *int which*) → unsigned int

tagged_stream_align_sptr.**set_min_noutput_items**(*tagged_stream_align_sptr self*, *int m*)

tagged_stream_align_sptr.**set_thread_priority**(*tagged_stream_align_sptr self*, *int priority*) → int

tagged_stream_align_sptr.**thread_priority**(*tagged_stream_align_sptr self*) → int

gnuradio.blocks.**tagged_stream_multiply_length**(*size_t itemsize*, *std::string const & lengthtagname*, *double scalar*) → tagged_stream_multiply_length_sptr

Allows scaling of a tagged stream length tag.

Searches for a specific tagged stream length tag and multiplies that length by a constant - for constant rate change blocks in a tagged stream

Constructor Specific Documentation:

Make a tagged stream multiply_length block.

Parameters:
- **itemsize** – Items size (number of bytes per item)
- **lengthtagname** – Length tag key
- **scalar** – value to scale length tag values by

tagged_stream_multiply_length_sptr.**active_thread_priority**(*tagged_stream_multiply_length_sptr self*) → int

tagged_stream_multiply_length_sptr.**declare_sample_delay**(*tagged_stream_multiply_length_sptr self*, *int which*, *int delay*)

declare_sample_delay(tagged_stream_multiply_length_sptr self, unsigned int delay)

tagged_stream_multiply_length_sptr.**message_subscribers**(*tagged_stream_multiply_length_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tagged_stream_multiply_length_sptr.**min_noutput_items**(*tagged_stream_multiply_length_sptr self*) → int

tagged_stream_multiply_length_sptr.**pc_input_buffers_full_avg**(*tagged_stream_multiply_length_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tagged_stream_multiply_length_sptr self) -> pmt_vector_float

tagged_stream_multiply_length_sptr.**pc_noutput_items_avg**(*tagged_stream_multiply_length_sptr self*) → float

tagged_stream_multiply_length_sptr.**pc_nproduced_avg**(*tagged_stream_multiply_length_sptr self*) → float

tagged_stream_multiply_length_sptr.**pc_output_buffers_full_avg**(*tagged_stream_multiply_length_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tagged_stream_multiply_length_sptr self) -> pmt_vector_float

tagged_stream_multiply_length_sptr.**pc_throughput_avg**(*tagged_stream_multiply_length_sptr self*) → float

tagged_stream_multiply_length_sptr.**pc_work_time_avg**(*tagged_stream_multiply_length_sptr self*) → float

tagged_stream_multiply_length_sptr.**pc_work_time_total**(*tagged_stream_multiply_length_sptr self*) → float

tagged_stream_multiply_length_sptr.**sample_delay**(*tagged_stream_multiply_length_sptr self*, *int which*) → unsigned int

tagged_stream_multiply_length_sptr.**set_min_noutput_items**(*tagged_stream_multiply_length_sptr self*, *int m*)

tagged_stream_multiply_length_sptr.**set_scalar**(*tagged_stream_multiply_length_sptr self*, *double scalar*)

tagged_stream_multiply_length_sptr.**set_thread_priority**(*tagged_stream_multiply_length_sptr self*, *int priority*) → int

tagged_stream_multiply_length_sptr.**thread_priority**(*tagged_stream_multiply_length_sptr self*) → int

gnuradio.blocks.**tagged_stream_mux**(*size_t itemsize*, *std::string const & lengthtagname*, *unsigned int tag_preserve_head_pos=0*) → tagged_stream_mux_sptr

Combines tagged streams.

Takes N streams as input. Each stream is tagged with packet lengths. Packets are output sequentially from each input stream.

The output signal has a new length tag, which is the sum of all individual length tags. The old length tags are discarded.

All other tags are propagated as expected, i.e. they stay associated with the same input item. There are cases when this behaviour is undesirable. One special case is when a tag at the first element (the head item) of one input port must stay on the head item of the output port. To achieve this, set to the port that will receive these special tags.

Constructor Specific Documentation:

Make a tagged stream mux block.

Parameters:
- **itemsize** – Items size (number of bytes per item)
- **lengthtagname** – Length tag key
- **tag_preserve_head_pos** – Preserves the head position of tags on this input port

tagged_stream_mux_sptr.**active_thread_priority**(*tagged_stream_mux_sptr self*) → int

tagged_stream_mux_sptr.**declare_sample_delay**(*tagged_stream_mux_sptr self*, *int which*, *int delay*)

declare_sample_delay(tagged_stream_mux_sptr self, unsigned int delay)

tagged_stream_mux_sptr.**message_subscribers**(*tagged_stream_mux_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tagged_stream_mux_sptr.**min_noutput_items**(*tagged_stream_mux_sptr self*) → int

tagged_stream_mux_sptr.**pc_input_buffers_full_avg**(*tagged_stream_mux_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tagged_stream_mux_sptr self) -> pmt_vector_float

tagged_stream_mux_sptr.**pc_noutput_items_avg**(*tagged_stream_mux_sptr self*) → float

tagged_stream_mux_sptr.**pc_nproduced_avg**(*tagged_stream_mux_sptr self*) → float

tagged_stream_mux_sptr.**pc_output_buffers_full_avg**(*tagged_stream_mux_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tagged_stream_mux_sptr self) -> pmt_vector_float

tagged_stream_mux_sptr.**pc_throughput_avg**(*tagged_stream_mux_sptr self*) → float

tagged_stream_mux_sptr.**pc_work_time_avg**(*tagged_stream_mux_sptr self*) → float

tagged_stream_mux_sptr.**pc_work_time_total**(*tagged_stream_mux_sptr self*) → float

tagged_stream_mux_sptr.**sample_delay**(*tagged_stream_mux_sptr self*, *int which*) → unsigned int

tagged_stream_mux_sptr.**set_min_noutput_items**(*tagged_stream_mux_sptr self*, *int m*)

tagged_stream_mux_sptr.**set_thread_priority**(*tagged_stream_mux_sptr self*, *int priority*) → int

tagged_stream_mux_sptr.**thread_priority**(*tagged_stream_mux_sptr self*) → int

gnuradio.blocks.**tagged_stream_to_pdu**(*gr::blocks::pdu::vector_type type*, *std::string const & lengthtagname*) → tagged_stream_to_pdu_sptr

Turns received stream data and tags into PDUs and sends them through a message port.

The sent message is a PMT-pair (created by pmt::cons()). The first element is a dictionary containing all the tags. The second is a vector containing the actual data.

Constructor Specific Documentation:

Construct a tagged_stream_to_pdu block.

Parameters:
- **type** – PDU type of pdu::vector_type
- **lengthtagname** – The name of the tag that specifies how long the packet is.

tagged_stream_to_pdu_sptr.**active_thread_priority**(*tagged_stream_to_pdu_sptr self*) → int

tagged_stream_to_pdu_sptr.**declare_sample_delay**(*tagged_stream_to_pdu_sptr self*, *int which*, *int delay*)

declare_sample_delay(tagged_stream_to_pdu_sptr self, unsigned int delay)

tagged_stream_to_pdu_sptr.**message_subscribers**(*tagged_stream_to_pdu_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tagged_stream_to_pdu_sptr.**min_noutput_items**(*tagged_stream_to_pdu_sptr self*) → int

tagged_stream_to_pdu_sptr.**pc_input_buffers_full_avg**(*tagged_stream_to_pdu_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tagged_stream_to_pdu_sptr self) -> pmt_vector_float

tagged_stream_to_pdu_sptr.**pc_noutput_items_avg**(*tagged_stream_to_pdu_sptr self*) → float

tagged_stream_to_pdu_sptr.**pc_nproduced_avg**(*tagged_stream_to_pdu_sptr self*) → float

tagged_stream_to_pdu_sptr.**pc_output_buffers_full_avg**(*tagged_stream_to_pdu_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tagged_stream_to_pdu_sptr self) -> pmt_vector_float

tagged_stream_to_pdu_sptr.**pc_throughput_avg**(*tagged_stream_to_pdu_sptr self*) → float

tagged_stream_to_pdu_sptr.**pc_work_time_avg**(*tagged_stream_to_pdu_sptr self*) → float

tagged_stream_to_pdu_sptr.**pc_work_time_total**(*tagged_stream_to_pdu_sptr self*) → float

tagged_stream_to_pdu_sptr.**sample_delay**(*tagged_stream_to_pdu_sptr self*, *int which*) → unsigned int

tagged_stream_to_pdu_sptr.**set_min_noutput_items**(*tagged_stream_to_pdu_sptr self*, *int m*)

tagged_stream_to_pdu_sptr.**set_thread_priority**(*tagged_stream_to_pdu_sptr self*, *int priority*) → int

tagged_stream_to_pdu_sptr.**thread_priority**(*tagged_stream_to_pdu_sptr self*) → int

gnuradio.blocks.**tags_strobe**(*size_t sizeof_stream_item*, *swig_int_ptr value*, *uint64_t nsamps*, *swig_int_ptr key*) → tags_strobe_sptr

Send tags at defined interval.

Sends a tag with key ‘strobe’ and a user-defined value (as a PMT) everynumber of samples. Useful for testing/debugging the tags system.

Because tags are sent with a data stream, this is a source block that acts identical to a null_source block.

Constructor Specific Documentation:

Make a tags stobe block to send tags with value every number of samples.

Parameters:
- **sizeof_stream_item** – size of the stream items in bytes.
- **value** – The value of the tags to send, as a PMT.
- **nsamps** – the number of samples between each tag.
- **key** – The tag key to sent

tags_strobe_sptr.**active_thread_priority**(*tags_strobe_sptr self*) → int

tags_strobe_sptr.**declare_sample_delay**(*tags_strobe_sptr self*, *int which*, *int delay*)

declare_sample_delay(tags_strobe_sptr self, unsigned int delay)

tags_strobe_sptr.**key**(*tags_strobe_sptr self*) → swig_int_ptr

Get the key of the tags being sent.

tags_strobe_sptr.**message_subscribers**(*tags_strobe_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tags_strobe_sptr.**min_noutput_items**(*tags_strobe_sptr self*) → int

tags_strobe_sptr.**nsamps**(*tags_strobe_sptr self*) → uint64_t

Get the number of samples between the tag strobe.

tags_strobe_sptr.**pc_input_buffers_full_avg**(*tags_strobe_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tags_strobe_sptr self) -> pmt_vector_float

tags_strobe_sptr.**pc_noutput_items_avg**(*tags_strobe_sptr self*) → float

tags_strobe_sptr.**pc_nproduced_avg**(*tags_strobe_sptr self*) → float

tags_strobe_sptr.**pc_output_buffers_full_avg**(*tags_strobe_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tags_strobe_sptr self) -> pmt_vector_float

tags_strobe_sptr.**pc_throughput_avg**(*tags_strobe_sptr self*) → float

tags_strobe_sptr.**pc_work_time_avg**(*tags_strobe_sptr self*) → float

tags_strobe_sptr.**pc_work_time_total**(*tags_strobe_sptr self*) → float

tags_strobe_sptr.**sample_delay**(*tags_strobe_sptr self*, *int which*) → unsigned int

tags_strobe_sptr.**set_key**(*tags_strobe_sptr self*, *swig_int_ptr key*)

Change the tag key we are sending

tags_strobe_sptr.**set_min_noutput_items**(*tags_strobe_sptr self*, *int m*)

tags_strobe_sptr.**set_nsamps**(*tags_strobe_sptr self*, *uint64_t nsamps*)

Reset the sending interval.

tags_strobe_sptr.**set_thread_priority**(*tags_strobe_sptr self*, *int priority*) → int

tags_strobe_sptr.**set_value**(*tags_strobe_sptr self*, *swig_int_ptr value*)

Reset the value of the tags being sent.

tags_strobe_sptr.**thread_priority**(*tags_strobe_sptr self*) → int

tags_strobe_sptr.**value**(*tags_strobe_sptr self*) → swig_int_ptr

Get the value of the tags being sent.

gnuradio.blocks.**tcp_server_sink**(*size_t itemsize*, *std::string const & host*, *int port*, *bool noblock=False*) → tcp_server_sink_sptr

Send stream trought an TCP socket.

Listen for incoming TCP connection(s). Duplicate data for each opened connection. Optionally can wait until first client connects before streaming starts.

Constructor Specific Documentation:

TCP Server Sink Constructor.

Parameters:

- **itemsize** – The size (in bytes) of the item datatype
- **host** – The name or IP address of interface to bind to.
- **port** – Port where to listen.
- **noblock** – If false, wait until first client connects before streaming starts. In non blocking mode (noblock=true), drop data onto floor if no client is connected.

tcp_server_sink_sptr.**active_thread_priority**(*tcp_server_sink_sptr self*) → int

tcp_server_sink_sptr.**declare_sample_delay**(*tcp_server_sink_sptr self*, *int which*, *int delay*)

declare_sample_delay(tcp_server_sink_sptr self, unsigned int delay)

tcp_server_sink_sptr.**message_subscribers**(*tcp_server_sink_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tcp_server_sink_sptr.**min_noutput_items**(*tcp_server_sink_sptr self*) → int

tcp_server_sink_sptr.**pc_input_buffers_full_avg**(*tcp_server_sink_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tcp_server_sink_sptr self) -> pmt_vector_float

tcp_server_sink_sptr.**pc_noutput_items_avg**(*tcp_server_sink_sptr self*) → float

tcp_server_sink_sptr.**pc_nproduced_avg**(*tcp_server_sink_sptr self*) → float

tcp_server_sink_sptr.**pc_output_buffers_full_avg**(*tcp_server_sink_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tcp_server_sink_sptr self) -> pmt_vector_float

tcp_server_sink_sptr.**pc_throughput_avg**(*tcp_server_sink_sptr self*) → float

tcp_server_sink_sptr.**pc_work_time_avg**(*tcp_server_sink_sptr self*) → float

tcp_server_sink_sptr.**pc_work_time_total**(*tcp_server_sink_sptr self*) → float

tcp_server_sink_sptr.**sample_delay**(*tcp_server_sink_sptr self*, *int which*) → unsigned int

tcp_server_sink_sptr.**set_min_noutput_items**(*tcp_server_sink_sptr self*, *int m*)

tcp_server_sink_sptr.**set_thread_priority**(*tcp_server_sink_sptr self*, *int priority*) → int

tcp_server_sink_sptr.**thread_priority**(*tcp_server_sink_sptr self*) → int

gnuradio.blocks.**threshold_ff**(*float lo*, *float hi*, *float initial_state=0*) → threshold_ff_sptr

Output a 1 or zero based on a threshold value.

Test the incoming signal against a threshold. If the signal excedes the value, it will output a 1 until the signal falls below the value.

Constructor Specific Documentation:

Parameters:
- **lo** –
- **hi** –
- **initial_state** –

threshold_ff_sptr.**active_thread_priority**(*threshold_ff_sptr self*) → int

threshold_ff_sptr.**declare_sample_delay**(*threshold_ff_sptr self*, *int which*, *int delay*)

declare_sample_delay(threshold_ff_sptr self, unsigned int delay)

threshold_ff_sptr.**hi**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**last_state**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**lo**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**message_subscribers**(*threshold_ff_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

threshold_ff_sptr.**min_noutput_items**(*threshold_ff_sptr self*) → int

threshold_ff_sptr.**pc_input_buffers_full_avg**(*threshold_ff_sptr self*, *int which*) → float

pc_input_buffers_full_avg(threshold_ff_sptr self) -> pmt_vector_float

threshold_ff_sptr.**pc_noutput_items_avg**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**pc_nproduced_avg**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**pc_output_buffers_full_avg**(*threshold_ff_sptr self*, *int which*) → float

pc_output_buffers_full_avg(threshold_ff_sptr self) -> pmt_vector_float

threshold_ff_sptr.**pc_throughput_avg**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**pc_work_time_avg**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**pc_work_time_total**(*threshold_ff_sptr self*) → float

threshold_ff_sptr.**sample_delay**(*threshold_ff_sptr self*, *int which*) → unsigned int

threshold_ff_sptr.**set_hi**(*threshold_ff_sptr self*, *float hi*)

threshold_ff_sptr.**set_last_state**(*threshold_ff_sptr self*, *float last_state*)

threshold_ff_sptr.**set_lo**(*threshold_ff_sptr self*, *float lo*)

threshold_ff_sptr.**set_min_noutput_items**(*threshold_ff_sptr self*, *int m*)

threshold_ff_sptr.**set_thread_priority**(*threshold_ff_sptr self*, *int priority*) → int

threshold_ff_sptr.**thread_priority**(*threshold_ff_sptr self*) → int

gnuradio.blocks.**throttle**(*size_t itemsize*, *double samples_per_sec*, *bool ignore_tags=True*) → throttle_sptr

throttle flow of samples such that the average rate does not exceed samples_per_sec.

input: one stream of itemsize; output: one stream of itemsize

N.B. this should only be used in GUI apps where there is no other rate limiting block. It is not intended nor effective at precisely controlling the rate of samples. That should be controlled by a source or sink tied to sample clock. E.g., a USRP or audio card.

Constructor Specific Documentation:

Parameters:
- **itemsize** –
- **samples_per_sec** –
- **ignore_tags** –

throttle_sptr.**active_thread_priority**(*throttle_sptr self*) → int

throttle_sptr.**declare_sample_delay**(*throttle_sptr self*, *int which*, *int delay*)

declare_sample_delay(throttle_sptr self, unsigned int delay)

throttle_sptr.**message_subscribers**(*throttle_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

throttle_sptr.**min_noutput_items**(*throttle_sptr self*) → int

throttle_sptr.**pc_input_buffers_full_avg**(*throttle_sptr self*, *int which*) → float

pc_input_buffers_full_avg(throttle_sptr self) -> pmt_vector_float

throttle_sptr.**pc_noutput_items_avg**(*throttle_sptr self*) → float

throttle_sptr.**pc_nproduced_avg**(*throttle_sptr self*) → float

throttle_sptr.**pc_output_buffers_full_avg**(*throttle_sptr self*, *int which*) → float

pc_output_buffers_full_avg(throttle_sptr self) -> pmt_vector_float

throttle_sptr.**pc_throughput_avg**(*throttle_sptr self*) → float

throttle_sptr.**pc_work_time_avg**(*throttle_sptr self*) → float

throttle_sptr.**pc_work_time_total**(*throttle_sptr self*) → float

throttle_sptr.**sample_delay**(*throttle_sptr self*, *int which*) → unsigned int

throttle_sptr.**sample_rate**(*throttle_sptr self*) → double

Get the sample rate in samples per second.

throttle_sptr.**set_min_noutput_items**(*throttle_sptr self*, *int m*)

throttle_sptr.**set_sample_rate**(*throttle_sptr self*, *double rate*)

Sets the sample rate in samples per second.

throttle_sptr.**set_thread_priority**(*throttle_sptr self*, *int priority*) → int

throttle_sptr.**thread_priority**(*throttle_sptr self*) → int

gnuradio.blocks.**transcendental**(*std::string const & name*, *std::string const & type*) → transcendental_sptr

A block that performs various transcendental math operations.

Possible function names can be found in the cmath library. IO may be either complex or real, double or single precision.

Possible type strings: float, double, complex_float, complex_double

output[i] = trans_fcn(input[i])

Constructor Specific Documentation:

Parameters:
- **name** –
- **type** –

transcendental_sptr.**active_thread_priority**(*transcendental_sptr self*) → int

transcendental_sptr.**declare_sample_delay**(*transcendental_sptr self*, *int which*, *int delay*)

declare_sample_delay(transcendental_sptr self, unsigned int delay)

transcendental_sptr.**message_subscribers**(*transcendental_sptr self*, *swig_int_ptr which_port*) →

swig_int_ptr

transcendental_sptr.**min_noutput_items**(*transcendental_sptr self*) → int

transcendental_sptr.**pc_input_buffers_full_avg**(*transcendental_sptr self*, *int which*) → float

pc_input_buffers_full_avg(transcendental_sptr self) -> pmt_vector_float

transcendental_sptr.**pc_noutput_items_avg**(*transcendental_sptr self*) → float

transcendental_sptr.**pc_nproduced_avg**(*transcendental_sptr self*) → float

transcendental_sptr.**pc_output_buffers_full_avg**(*transcendental_sptr self*, *int which*) → float

pc_output_buffers_full_avg(transcendental_sptr self) -> pmt_vector_float

transcendental_sptr.**pc_throughput_avg**(*transcendental_sptr self*) → float

transcendental_sptr.**pc_work_time_avg**(*transcendental_sptr self*) → float

transcendental_sptr.**pc_work_time_total**(*transcendental_sptr self*) → float

transcendental_sptr.**sample_delay**(*transcendental_sptr self*, *int which*) → unsigned int

transcendental_sptr.**set_min_noutput_items**(*transcendental_sptr self*, *int m*)

transcendental_sptr.**set_thread_priority**(*transcendental_sptr self*, *int priority*) → int

transcendental_sptr.**thread_priority**(*transcendental_sptr self*) → int

gnuradio.blocks.**tsb_vector_sink_b**(*int vlen=1*, *std::string const & tsb_key*) → tsb_vector_sink_b_sptr

A vector sink for tagged streams.

Unlike a gr::blocks::vector_sink_f, this only works with tagged streams.

Constructor Specific Documentation:

Parameters:
- **vlen** – Vector length
- **tsb_key** – Tagged Stream Key

tsb_vector_sink_b_sptr.**active_thread_priority**(*tsb_vector_sink_b_sptr self*) → int

tsb_vector_sink_b_sptr.**data**(*tsb_vector_sink_b_sptr self*) → std::vector< std::vector< unsigned char,std::allocator< unsigned char > >,std::allocator< std::vector< unsigned char,std::allocator< unsigned char > > > >

tsb_vector_sink_b_sptr.**declare_sample_delay**(*tsb_vector_sink_b_sptr self*, *int which*, *int delay*)

declare_sample_delay(tsb_vector_sink_b_sptr self, unsigned int delay)

tsb_vector_sink_b_sptr.**message_subscribers**(*tsb_vector_sink_b_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tsb_vector_sink_b_sptr.**min_noutput_items**(*tsb_vector_sink_b_sptr self*) → int

tsb_vector_sink_b_sptr.**pc_input_buffers_full_avg**(*tsb_vector_sink_b_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tsb_vector_sink_b_sptr self) -> pmt_vector_float

tsb_vector_sink_b_sptr.**pc_noutput_items_avg**(*tsb_vector_sink_b_sptr self*) → float

tsb_vector_sink_b_sptr.**pc_nproduced_avg**(*tsb_vector_sink_b_sptr self*) → float

tsb_vector_sink_b_sptr.**pc_output_buffers_full_avg**(*tsb_vector_sink_b_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tsb_vector_sink_b_sptr self) -> pmt_vector_float

tsb_vector_sink_b_sptr.**pc_throughput_avg**(*tsb_vector_sink_b_sptr self*) → float

tsb_vector_sink_b_sptr.**pc_work_time_avg**(*tsb_vector_sink_b_sptr self*) → float

tsb_vector_sink_b_sptr.**pc_work_time_total**(*tsb_vector_sink_b_sptr self*) → float

tsb_vector_sink_b_sptr.**reset**(*tsb_vector_sink_b_sptr self*)

tsb_vector_sink_b_sptr.**sample_delay**(*tsb_vector_sink_b_sptr self*, *int which*) → unsigned int

tsb_vector_sink_b_sptr.**set_min_noutput_items**(*tsb_vector_sink_b_sptr self*, *int m*)

tsb_vector_sink_b_sptr.**set_thread_priority**(*tsb_vector_sink_b_sptr self*, *int priority*) → int

tsb_vector_sink_b_sptr.**tags**(*tsb_vector_sink_b_sptr self*) → tags_vector_t

Doesn't include the TSB tags.

tsb_vector_sink_b_sptr.**thread_priority**(*tsb_vector_sink_b_sptr self*) → int

gnuradio.blocks.**tsb_vector_sink_c**(*int vlen=1*, *std::string const & tsb_key*) → tsb_vector_sink_c_sptr

A vector sink for tagged streams.

Unlike a gr::blocks::vector_sink_f, this only works with tagged streams.

Constructor Specific Documentation:

Parameters:
- **vlen** – Vector length
- **tsb_key** – Tagged Stream Key

tsb_vector_sink_c_sptr.**active_thread_priority**(*tsb_vector_sink_c_sptr self*) → int

tsb_vector_sink_c_sptr.**data**(*tsb_vector_sink_c_sptr self*) → gr_vector_vector_complexf

tsb_vector_sink_c_sptr.**declare_sample_delay**(*tsb_vector_sink_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(tsb_vector_sink_c_sptr self, unsigned int delay)

tsb_vector_sink_c_sptr.**message_subscribers**(*tsb_vector_sink_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tsb_vector_sink_c_sptr.**min_noutput_items**(*tsb_vector_sink_c_sptr self*) → int

tsb_vector_sink_c_sptr.**pc_input_buffers_full_avg**(*tsb_vector_sink_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tsb_vector_sink_c_sptr self) -> pmt_vector_float

tsb_vector_sink_c_sptr.**pc_noutput_items_avg**(*tsb_vector_sink_c_sptr self*) → float

tsb_vector_sink_c_sptr.**pc_nproduced_avg**(*tsb_vector_sink_c_sptr self*) → float

tsb_vector_sink_c_sptr.**pc_output_buffers_full_avg**(*tsb_vector_sink_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tsb_vector_sink_c_sptr self) -> pmt_vector_float

tsb_vector_sink_c_sptr.**pc_throughput_avg**(*tsb_vector_sink_c_sptr self*) → float

tsb_vector_sink_c_sptr.**pc_work_time_avg**(*tsb_vector_sink_c_sptr self*) → float

tsb_vector_sink_c_sptr.**pc_work_time_total**(*tsb_vector_sink_c_sptr self*) → float

tsb_vector_sink_c_sptr.**reset**(*tsb_vector_sink_c_sptr self*)

tsb_vector_sink_c_sptr.**sample_delay**(*tsb_vector_sink_c_sptr self*, *int which*) → unsigned int

tsb_vector_sink_c_sptr.**set_min_noutput_items**(*tsb_vector_sink_c_sptr self*, *int m*)

tsb_vector_sink_c_sptr.**set_thread_priority**(*tsb_vector_sink_c_sptr self*, *int priority*) → int

tsb_vector_sink_c_sptr.**tags**(*tsb_vector_sink_c_sptr self*) → tags_vector_t

Doesn't include the TSB tags.

tsb_vector_sink_c_sptr.**thread_priority**(*tsb_vector_sink_c_sptr self*) → int

gnuradio.blocks.**tsb_vector_sink_f**(*int vlen=1*, *std::string const & tsb_key*) → tsb_vector_sink_f_sptr

A vector sink for tagged streams.

Unlike a gr::blocks::vector_sink_f, this only works with tagged streams.

Constructor Specific Documentation:

Parameters:
- **vlen** – Vector length
- **tsb_key** – Tagged Stream Key

tsb_vector_sink_f_sptr.**active_thread_priority**(*tsb_vector_sink_f_sptr self*) → int

tsb_vector_sink_f_sptr.**data**(*tsb_vector_sink_f_sptr self*) → std::vector< std::vector< float,std::allocator< float > >,std::allocator< std::vector< float,std::allocator< float > > > >

tsb_vector_sink_f_sptr.**declare_sample_delay**(*tsb_vector_sink_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(tsb_vector_sink_f_sptr self, unsigned int delay)

tsb_vector_sink_f_sptr.**message_subscribers**(*tsb_vector_sink_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tsb_vector_sink_f_sptr.**min_noutput_items**(*tsb_vector_sink_f_sptr self*) → int

tsb_vector_sink_f_sptr.**pc_input_buffers_full_avg**(*tsb_vector_sink_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tsb_vector_sink_f_sptr self) -> pmt_vector_float

tsb_vector_sink_f_sptr.**pc_noutput_items_avg**(*tsb_vector_sink_f_sptr self*) → float

tsb_vector_sink_f_sptr.**pc_nproduced_avg**(*tsb_vector_sink_f_sptr self*) → float

tsb_vector_sink_f_sptr.**pc_output_buffers_full_avg**(*tsb_vector_sink_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tsb_vector_sink_f_sptr self) -> pmt_vector_float

tsb_vector_sink_f_sptr.**pc_throughput_avg**(*tsb_vector_sink_f_sptr self*) → float

tsb_vector_sink_f_sptr.**pc_work_time_avg**(*tsb_vector_sink_f_sptr self*) → float

tsb_vector_sink_f_sptr.**pc_work_time_total**(*tsb_vector_sink_f_sptr self*) → float

tsb_vector_sink_f_sptr.**reset**(*tsb_vector_sink_f_sptr self*)

tsb_vector_sink_f_sptr.**sample_delay**(*tsb_vector_sink_f_sptr self*, *int which*) → unsigned int

tsb_vector_sink_f_sptr.**set_min_noutput_items**(*tsb_vector_sink_f_sptr self*, *int m*)

tsb_vector_sink_f_sptr.**set_thread_priority**(*tsb_vector_sink_f_sptr self*, *int priority*) → int

tsb_vector_sink_f_sptr.**tags**(*tsb_vector_sink_f_sptr self*) → tags_vector_t

Doesn't include the TSB tags.

tsb_vector_sink_f_sptr.**thread_priority**(*tsb_vector_sink_f_sptr self*) → int

gnuradio.blocks.**tsb_vector_sink_i**(*int vlen=1*, *std::string const & tsb_key*) → tsb_vector_sink_i_sptr

A vector sink for tagged streams.

Unlike a gr::blocks::vector_sink_f, this only works with tagged streams.

Constructor Specific Documentation:

Parameters:
- **vlen** – Vector length
- **tsb_key** – Tagged Stream Key

tsb_vector_sink_i_sptr.**active_thread_priority**(*tsb_vector_sink_i_sptr self*) → int

tsb_vector_sink_i_sptr.**data**(*tsb_vector_sink_i_sptr self*) → std::vector< std::vector< int,std::allocator< int > >,std::allocator< std::vector< int,std::allocator< int > > > >

tsb_vector_sink_i_sptr.**declare_sample_delay**(*tsb_vector_sink_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(tsb_vector_sink_i_sptr self, unsigned int delay)

tsb_vector_sink_i_sptr.**message_subscribers**(*tsb_vector_sink_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tsb_vector_sink_i_sptr.**min_noutput_items**(*tsb_vector_sink_i_sptr self*) → int

tsb_vector_sink_i_sptr.**pc_input_buffers_full_avg**(*tsb_vector_sink_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tsb_vector_sink_i_sptr self) -> pmt_vector_float

tsb_vector_sink_i_sptr.**pc_noutput_items_avg**(*tsb_vector_sink_i_sptr self*) → float

tsb_vector_sink_i_sptr.**pc_nproduced_avg**(*tsb_vector_sink_i_sptr self*) → float

tsb_vector_sink_i_sptr.**pc_output_buffers_full_avg**(*tsb_vector_sink_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tsb_vector_sink_i_sptr self) -> pmt_vector_float

tsb_vector_sink_i_sptr.**pc_throughput_avg**(*tsb_vector_sink_i_sptr self*) → float

tsb_vector_sink_i_sptr.**pc_work_time_avg**(*tsb_vector_sink_i_sptr self*) → float

tsb_vector_sink_i_sptr.**pc_work_time_total**(*tsb_vector_sink_i_sptr self*) → float

tsb_vector_sink_i_sptr.**reset**(*tsb_vector_sink_i_sptr self*)

tsb_vector_sink_i_sptr.**sample_delay**(*tsb_vector_sink_i_sptr self*, *int which*) → unsigned int

tsb_vector_sink_i_sptr.**set_min_noutput_items**(*tsb_vector_sink_i_sptr self*, *int m*)

tsb_vector_sink_i_sptr.**set_thread_priority**(*tsb_vector_sink_i_sptr self*, *int priority*) → int

tsb_vector_sink_i_sptr.**tags**(*tsb_vector_sink_i_sptr self*) → tags_vector_t

Doesn't include the TSB tags.

tsb_vector_sink_i_sptr.**thread_priority**(*tsb_vector_sink_i_sptr self*) → int

gnuradio.blocks.**tsb_vector_sink_s**(*int vlen=1*, *std::string const & tsb_key*) → tsb_vector_sink_s_sptr

A vector sink for tagged streams.

Unlike a gr::blocks::vector_sink_f, this only works with tagged streams.

Constructor Specific Documentation:

Parameters:
- **vlen** – Vector length
- **tsb_key** – Tagged Stream Key

tsb_vector_sink_s_sptr.**active_thread_priority**(*tsb_vector_sink_s_sptr self*) → int

tsb_vector_sink_s_sptr.**data**(*tsb_vector_sink_s_sptr self*) → std::vector< std::vector< short,std::allocator< short > >,std::allocator< std::vector< short,std::allocator< short > > > >

tsb_vector_sink_s_sptr.**declare_sample_delay**(*tsb_vector_sink_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(tsb_vector_sink_s_sptr self, unsigned int delay)

tsb_vector_sink_s_sptr.**message_subscribers**(*tsb_vector_sink_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tsb_vector_sink_s_sptr.**min_noutput_items**(*tsb_vector_sink_s_sptr self*) → int

tsb_vector_sink_s_sptr.**pc_input_buffers_full_avg**(*tsb_vector_sink_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tsb_vector_sink_s_sptr self) -> pmt_vector_float

tsb_vector_sink_s_sptr.**pc_noutput_items_avg**(*tsb_vector_sink_s_sptr self*) → float

tsb_vector_sink_s_sptr.**pc_nproduced_avg**(*tsb_vector_sink_s_sptr self*) → float

tsb_vector_sink_s_sptr.**pc_output_buffers_full_avg**(*tsb_vector_sink_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tsb_vector_sink_s_sptr self) -> pmt_vector_float

tsb_vector_sink_s_sptr.**pc_throughput_avg**(*tsb_vector_sink_s_sptr self*) → float

tsb_vector_sink_s_sptr.**pc_work_time_avg**(*tsb_vector_sink_s_sptr self*) → float

tsb_vector_sink_s_sptr.**pc_work_time_total**(*tsb_vector_sink_s_sptr self*) → float

tsb_vector_sink_s_sptr.**reset**(*tsb_vector_sink_s_sptr self*)

tsb_vector_sink_s_sptr.**sample_delay**(*tsb_vector_sink_s_sptr self*, *int which*) → unsigned int

tsb_vector_sink_s_sptr.**set_min_noutput_items**(*tsb_vector_sink_s_sptr self*, *int m*)

tsb_vector_sink_s_sptr.**set_thread_priority**(*tsb_vector_sink_s_sptr self*, *int priority*) → int

tsb_vector_sink_s_sptr.**tags**(*tsb_vector_sink_s_sptr self*) → tags_vector_t

Doesn't include the TSB tags.

tsb_vector_sink_s_sptr.**thread_priority**(*tsb_vector_sink_s_sptr self*) → int

gnuradio.blocks.**tuntap_pdu**(*std::string dev*, *int MTU=10000*, *bool istunflag=False*) → tuntap_pdu_sptr

Creates TUNTAP interface and translates traffic to PDUs.

Constructor Specific Documentation:

Construct a TUNTAP PDU interface.

Parameters:
- **dev** – Device name to create
- **MTU** – Maximum Transmission Unit size
- **istunflag** – Flag to indicate TUN or Tap

tuntap_pdu_sptr.**active_thread_priority**(*tuntap_pdu_sptr self*) → int

tuntap_pdu_sptr.**declare_sample_delay**(*tuntap_pdu_sptr self*, *int which*, *int delay*)

declare_sample_delay(tuntap_pdu_sptr self, unsigned int delay)

tuntap_pdu_sptr.**message_subscribers**(*tuntap_pdu_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

tuntap_pdu_sptr.**min_noutput_items**(*tuntap_pdu_sptr self*) → int

tuntap_pdu_sptr.**pc_input_buffers_full_avg**(*tuntap_pdu_sptr self*, *int which*) → float

pc_input_buffers_full_avg(tuntap_pdu_sptr self) -> pmt_vector_float

tuntap_pdu_sptr.**pc_noutput_items_avg**(*tuntap_pdu_sptr self*) → float

tuntap_pdu_sptr.**pc_nproduced_avg**(*tuntap_pdu_sptr self*) → float

tuntap_pdu_sptr.**pc_output_buffers_full_avg**(*tuntap_pdu_sptr self*, *int which*) → float

pc_output_buffers_full_avg(tuntap_pdu_sptr self) -> pmt_vector_float

tuntap_pdu_sptr.**pc_throughput_avg**(*tuntap_pdu_sptr self*) → float

tuntap_pdu_sptr.**pc_work_time_avg**(*tuntap_pdu_sptr self*) → float

tuntap_pdu_sptr.**pc_work_time_total**(*tuntap_pdu_sptr self*) → float

tuntap_pdu_sptr.**sample_delay**(*tuntap_pdu_sptr self*, *int which*) → unsigned int

tuntap_pdu_sptr.**set_min_noutput_items**(*tuntap_pdu_sptr self*, *int m*)

tuntap_pdu_sptr.**set_thread_priority**(*tuntap_pdu_sptr self*, *int priority*) → int

tuntap_pdu_sptr.**thread_priority**(*tuntap_pdu_sptr self*) → int

gnuradio.blocks.**uchar_to_float**() → uchar_to_float_sptr

Convert stream of unsigned chars to a stream of floats.

Constructor Specific Documentation:

Build a uchar to float block.

uchar_to_float_sptr.**active_thread_priority**(*uchar_to_float_sptr self*) → int

uchar_to_float_sptr.**declare_sample_delay**(*uchar_to_float_sptr self*, *int which*, *int delay*)

declare_sample_delay(uchar_to_float_sptr self, unsigned int delay)

uchar_to_float_sptr.**message_subscribers**(*uchar_to_float_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

uchar_to_float_sptr.**min_noutput_items**(*uchar_to_float_sptr self*) → int

uchar_to_float_sptr.**pc_input_buffers_full_avg**(*uchar_to_float_sptr self*, *int which*) → float

pc_input_buffers_full_avg(uchar_to_float_sptr self) -> pmt_vector_float

uchar_to_float_sptr.**pc_noutput_items_avg**(*uchar_to_float_sptr self*) → float

uchar_to_float_sptr.**pc_nproduced_avg**(*uchar_to_float_sptr self*) → float

uchar_to_float_sptr.**pc_output_buffers_full_avg**(*uchar_to_float_sptr self*, *int which*) → float

pc_output_buffers_full_avg(uchar_to_float_sptr self) -> pmt_vector_float

uchar_to_float_sptr.**pc_throughput_avg**(*uchar_to_float_sptr self*) → float

uchar_to_float_sptr.**pc_work_time_avg**(*uchar_to_float_sptr self*) → float

uchar_to_float_sptr.**pc_work_time_total**(*uchar_to_float_sptr self*) → float

uchar_to_float_sptr.**sample_delay**(*uchar_to_float_sptr self*, *int which*) → unsigned int

uchar_to_float_sptr.**set_min_noutput_items**(*uchar_to_float_sptr self*, *int m*)

uchar_to_float_sptr.**set_thread_priority**(*uchar_to_float_sptr self*, *int priority*) → int

uchar_to_float_sptr.**thread_priority**(*uchar_to_float_sptr self*) → int

gnuradio.blocks.**udp_sink**(*size_t itemsize*, *std::string const & host*, *int port*, *int payload_size=1472*, *bool eof=True*) → udp_sink_sptr

Write stream to an UDP socket.

Constructor Specific Documentation:

UDP Sink Constructor.

Parameters:
- **itemsize** – The size (in bytes) of the item datatype
- **host** – The name or IP address of the receiving host; use NULL or None for no connection
- **port** – Destination port to connect to on receiving host
- **payload_size** – UDP payload size by default set to 1472 = (1500 MTU - (8 byte UDP header) - (20 byte IP header))
- **eof** – Send zero-length packet on disconnect

udp_sink_sptr.**active_thread_priority**(*udp_sink_sptr self*) → int

udp_sink_sptr.**connect**(*udp_sink_sptr self*, *std::string const & host*, *int port*)

Change the connection to a new destination.

Calls disconnect() to terminate any current connection first.

udp_sink_sptr.**declare_sample_delay**(*udp_sink_sptr self*, *int which*, *int delay*)

declare_sample_delay(udp_sink_sptr self, unsigned int delay)

udp_sink_sptr.**disconnect**(*udp_sink_sptr self*)

Send zero-length packet (if eof is requested) then stop sending.

Zero-byte packets can be interpreted as EOF by gr_udp_source. Note that disconnect occurs automatically when the sink is destroyed, but not when its top_block stops.

udp_sink_sptr.**message_subscribers**(*udp_sink_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

udp_sink_sptr.**min_noutput_items**(*udp_sink_sptr self*) → int

udp_sink_sptr.**payload_size**(*udp_sink_sptr self*) → int

return the PAYLOAD_SIZE of the socket

udp_sink_sptr.**pc_input_buffers_full_avg**(*udp_sink_sptr self*, *int which*) → float

pc_input_buffers_full_avg(udp_sink_sptr self) -> pmt_vector_float

udp_sink_sptr.**pc_noutput_items_avg**(*udp_sink_sptr self*) → float

udp_sink_sptr.**pc_nproduced_avg**(*udp_sink_sptr self*) → float

udp_sink_sptr.**pc_output_buffers_full_avg**(*udp_sink_sptr self*, *int which*) → float

pc_output_buffers_full_avg(udp_sink_sptr self) -> pmt_vector_float

udp_sink_sptr.**pc_throughput_avg**(*udp_sink_sptr self*) → float

udp_sink_sptr.**pc_work_time_avg**(*udp_sink_sptr self*) → float

udp_sink_sptr.**pc_work_time_total**(*udp_sink_sptr self*) → float

udp_sink_sptr.**sample_delay**(*udp_sink_sptr self*, *int which*) → unsigned int

udp_sink_sptr.**set_min_noutput_items**(*udp_sink_sptr self*, *int m*)

udp_sink_sptr.**set_thread_priority**(*udp_sink_sptr self*, *int priority*) → int

udp_sink_sptr.**thread_priority**(*udp_sink_sptr self*) → int

gnuradio.blocks.**udp_source**(*size_t itemsize*, *std::string const & host*, *int port*, *int payload_size=1472*, *bool eof=True*) → udp_source_sptr

Read stream from an UDP socket.

Constructor Specific Documentation:

UDP Source Constructor.

Parameters:
- **itemsize** – The size (in bytes) of the item datatype
- **host** – The name or IP address of the transmitting host; can be NULL, None, or “0.0.0.0” to allow reading from any interface on the host
- **port** – The port number on which to receive data; use 0 to have the system assign an unused port number
- **payload_size** – UDP payload size by default set to 1472 = (1500 MTU - (8 byte UDP header) - (20 byte IP header))
- **eof** – Interpret zero-length packet as EOF (default: true)

udp_source_sptr.**active_thread_priority**(*udp_source_sptr self*) → int

udp_source_sptr.**connect**(*udp_source_sptr self*, *std::string const & host*, *int port*)

Change the connection to a new destination.

Calls disconnect() to terminate any current connection first.

udp_source_sptr.**declare_sample_delay**(*udp_source_sptr self*, *int which*, *int delay*)
declare_sample_delay(udp_source_sptr self, unsigned int delay)

udp_source_sptr.**disconnect**(*udp_source_sptr self*)
Cut the connection if we have one set up.

udp_source_sptr.**get_port**(*udp_source_sptr self*) → int
return the port number of the socket

udp_source_sptr.**message_subscribers**(*udp_source_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

udp_source_sptr.**min_noutput_items**(*udp_source_sptr self*) → int

udp_source_sptr.**payload_size**(*udp_source_sptr self*) → int
return the PAYLOAD_SIZE of the socket

udp_source_sptr.**pc_input_buffers_full_avg**(*udp_source_sptr self*, *int which*) → float
pc_input_buffers_full_avg(udp_source_sptr self) -> pmt_vector_float

udp_source_sptr.**pc_noutput_items_avg**(*udp_source_sptr self*) → float

udp_source_sptr.**pc_nproduced_avg**(*udp_source_sptr self*) → float

udp_source_sptr.**pc_output_buffers_full_avg**(*udp_source_sptr self*, *int which*) → float
pc_output_buffers_full_avg(udp_source_sptr self) -> pmt_vector_float

udp_source_sptr.**pc_throughput_avg**(*udp_source_sptr self*) → float

udp_source_sptr.**pc_work_time_avg**(*udp_source_sptr self*) → float

udp_source_sptr.**pc_work_time_total**(*udp_source_sptr self*) → float

udp_source_sptr.**sample_delay**(*udp_source_sptr self*, *int which*) → unsigned int

udp_source_sptr.**set_min_noutput_items**(*udp_source_sptr self*, *int m*)

udp_source_sptr.**set_thread_priority**(*udp_source_sptr self*, *int priority*) → int

udp_source_sptr.**thread_priority**(*udp_source_sptr self*) → int

gnuradio.blocks.**unpack_k_bits_bb**(*unsigned int k*) → unpack_k_bits_bb_sptr
Converts a byte with k relevant bits to k output bytes with 1 bit in the LSB.

This block picks the K least significant bits from a byte, and expands them into K bytes of 0 or 1.

Example: k = 4 in = [0xf5, 0x08] out = [0,1,0,1, 1,0,0,0]

Constructor Specific Documentation:

Make an unpack_k_bits block.

Parameters: **k** – number of bits to unpack.

unpack_k_bits_bb_sptr.**active_thread_priority**(*unpack_k_bits_bb_sptr self*) → int

unpack_k_bits_bb_sptr.**declare_sample_delay**(*unpack_k_bits_bb_sptr self*, *int which*, *int delay*)
declare_sample_delay(unpack_k_bits_bb_sptr self, unsigned int delay)

unpack_k_bits_bb_sptr.**message_subscribers**(*unpack_k_bits_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

unpack_k_bits_bb_sptr.**min_noutput_items**(*unpack_k_bits_bb_sptr self*) → int

unpack_k_bits_bb_sptr.**pc_input_buffers_full_avg**(*unpack_k_bits_bb_sptr self*, *int which*) → float
pc_input_buffers_full_avg(unpack_k_bits_bb_sptr self) -> pmt_vector_float

unpack_k_bits_bb_sptr.**pc_noutput_items_avg**(*unpack_k_bits_bb_sptr self*) → float

unpack_k_bits_bb_sptr.**pc_nproduced_avg**(*unpack_k_bits_bb_sptr self*) → float

unpack_k_bits_bb_sptr.**pc_output_buffers_full_avg**(*unpack_k_bits_bb_sptr self*, *int which*) → float
pc_output_buffers_full_avg(unpack_k_bits_bb_sptr self) -> pmt_vector_float

unpack_k_bits_bb_sptr.**pc_throughput_avg**(*unpack_k_bits_bb_sptr self*) → float

unpack_k_bits_bb_sptr.**pc_work_time_avg**(*unpack_k_bits_bb_sptr self*) → float

unpack_k_bits_bb_sptr.**pc_work_time_total**(*unpack_k_bits_bb_sptr self*) → float

unpack_k_bits_bb_sptr.**sample_delay**(*unpack_k_bits_bb_sptr self*, *int which*) → unsigned int

unpack_k_bits_bb_sptr.**set_min_noutput_items**(*unpack_k_bits_bb_sptr self*, *int m*)

unpack_k_bits_bb_sptr.**set_thread_priority**(*unpack_k_bits_bb_sptr self*, *int priority*) → int

unpack_k_bits_bb_sptr.**thread_priority**(*unpack_k_bits_bb_sptr self*) → int

gnuradio.blocks.**unpacked_to_packed_bb**(*unsigned int bits_per_chunk*, *gr::endianness_t endianness*) → unpacked_to_packed_bb_sptr

Convert a stream of unpacked bytes or shorts into a stream of packed bytes or shorts.

input: stream of unsigned char; output: stream of unsigned char

This is the inverse of gr::blocks::packed_to_unpacked_XX.

The low bits are extracted from each input byte or short. These bits are then packed densely into the output bytes or shorts, such that all 8 or 16 bits of the output bytes or shorts are filled with valid input bits. The right thing is done if bits_per_chunk is not a power of two.

The combination of gr::blocks::packed_to_unpacked_XX followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols.

Constructor Specific Documentation:

Parameters:
- **bits_per_chunk** –
- **endianness** –

unpacked_to_packed_bb_sptr.**active_thread_priority**(*unpacked_to_packed_bb_sptr self*) → int

unpacked_to_packed_bb_sptr.**declare_sample_delay**(*unpacked_to_packed_bb_sptr self*, *int which*, *int delay*)

declare_sample_delay(unpacked_to_packed_bb_sptr self, unsigned int delay)

unpacked_to_packed_bb_sptr.**message_subscribers**(*unpacked_to_packed_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

unpacked_to_packed_bb_sptr.**min_noutput_items**(*unpacked_to_packed_bb_sptr self*) → int

unpacked_to_packed_bb_sptr.**pc_input_buffers_full_avg**(*unpacked_to_packed_bb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(unpacked_to_packed_bb_sptr self) -> pmt_vector_float

unpacked_to_packed_bb_sptr.**pc_noutput_items_avg**(*unpacked_to_packed_bb_sptr self*) → float

unpacked_to_packed_bb_sptr.**pc_nproduced_avg**(*unpacked_to_packed_bb_sptr self*) → float

unpacked_to_packed_bb_sptr.**pc_output_buffers_full_avg**(*unpacked_to_packed_bb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(unpacked_to_packed_bb_sptr self) -> pmt_vector_float

unpacked_to_packed_bb_sptr.**pc_throughput_avg**(*unpacked_to_packed_bb_sptr self*) → float

unpacked_to_packed_bb_sptr.**pc_work_time_avg**(*unpacked_to_packed_bb_sptr self*) → float

unpacked_to_packed_bb_sptr.**pc_work_time_total**(*unpacked_to_packed_bb_sptr self*) → float

unpacked_to_packed_bb_sptr.**sample_delay**(*unpacked_to_packed_bb_sptr self*, *int which*) → unsigned int

unpacked_to_packed_bb_sptr.**set_min_noutput_items**(*unpacked_to_packed_bb_sptr self*, *int m*)

unpacked_to_packed_bb_sptr.**set_thread_priority**(*unpacked_to_packed_bb_sptr self*, *int priority*) → int

unpacked_to_packed_bb_sptr.**thread_priority**(*unpacked_to_packed_bb_sptr self*) → int

gnuradio.blocks.**unpacked_to_packed_ii**(*unsigned int bits_per_chunk*, *gr::endianness_t endianness*) → unpacked_to_packed_ii_sptr

Convert a stream of unpacked bytes or shorts into a stream of packed bytes or shorts.

input: stream of int; output: stream of int

This is the inverse of gr::blocks::packed_to_unpacked_XX.

The low bits are extracted from each input byte or short. These bits are then packed densely into the output bytes or shorts, such that all 8 or 16 bits of the output bytes or shorts are filled with valid input bits. The right thing is done if bits_per_chunk is not a power of two.

The combination of gr::blocks::packed_to_unpacked_XX followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols.

Constructor Specific Documentation:

Parameters:
- **bits_per_chunk** –
- **endianness** –

unpacked_to_packed_ii_sptr.**active_thread_priority**(*unpacked_to_packed_ii_sptr self*) → int

unpacked_to_packed_ii_sptr.**declare_sample_delay**(*unpacked_to_packed_ii_sptr self*, *int which*, *int delay*)

declare_sample_delay(unpacked_to_packed_ii_sptr self, unsigned int delay)

unpacked_to_packed_ii_sptr.**message_subscribers**(*unpacked_to_packed_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

unpacked_to_packed_ii_sptr.**min_noutput_items**(*unpacked_to_packed_ii_sptr self*) → int

unpacked_to_packed_ii_sptr.**pc_input_buffers_full_avg**(*unpacked_to_packed_ii_sptr self*, *int which*) → float

pc_input_buffers_full_avg(unpacked_to_packed_ii_sptr self) -> pmt_vector_float

unpacked_to_packed_ii_sptr.**pc_noutput_items_avg**(*unpacked_to_packed_ii_sptr self*) → float

unpacked_to_packed_ii_sptr.**pc_nproduced_avg**(*unpacked_to_packed_ii_sptr self*) → float

unpacked_to_packed_ii_sptr.**pc_output_buffers_full_avg**(*unpacked_to_packed_ii_sptr self*, *int which*) → float

pc_output_buffers_full_avg(unpacked_to_packed_ii_sptr self) -> pmt_vector_float

unpacked_to_packed_ii_sptr.**pc_throughput_avg**(*unpacked_to_packed_ii_sptr self*) → float

unpacked_to_packed_ii_sptr.**pc_work_time_avg**(*unpacked_to_packed_ii_sptr self*) → float

unpacked_to_packed_ii_sptr.**pc_work_time_total**(*unpacked_to_packed_ii_sptr self*) → float

unpacked_to_packed_ii_sptr.**sample_delay**(*unpacked_to_packed_ii_sptr self*, *int which*) → unsigned int

unpacked_to_packed_ii_sptr.**set_min_noutput_items**(*unpacked_to_packed_ii_sptr self*, *int m*)

unpacked_to_packed_ii_sptr.**set_thread_priority**(*unpacked_to_packed_ii_sptr self*, *int priority*) → int

unpacked_to_packed_ii_sptr.**thread_priority**(*unpacked_to_packed_ii_sptr self*) → int

gnuradio.blocks.**unpacked_to_packed_ss**(*unsigned int bits_per_chunk*, *gr::endianness_t endianness*) → unpacked_to_packed_ss_sptr

Convert a stream of unpacked bytes or shorts into a stream of packed bytes or shorts.

input: stream of short; output: stream of short

This is the inverse of gr::blocks::packed_to_unpacked_XX.

The low bits are extracted from each input byte or short. These bits are then packed densely into the output bytes or shorts, such that all 8 or 16 bits of the output bytes or shorts are filled with valid input bits. The right thing is done if bits_per_chunk is not a power of two.

The combination of gr::blocks::packed_to_unpacked_XX followed by gr_chunks_to_symbols_Xf or gr_chunks_to_symbols_Xc handles the general case of mapping from a stream of bytes or shorts into arbitrary float or complex symbols.

Constructor Specific Documentation:

Parameters:
- **bits_per_chunk** –
- **endianness** –

unpacked_to_packed_ss_sptr.**active_thread_priority**(*unpacked_to_packed_ss_sptr self*) → int

unpacked_to_packed_ss_sptr.**declare_sample_delay**(*unpacked_to_packed_ss_sptr self*, *int which*, *int delay*)

declare_sample_delay(unpacked_to_packed_ss_sptr self, unsigned int delay)

unpacked_to_packed_ss_sptr.**message_subscribers**(*unpacked_to_packed_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

unpacked_to_packed_ss_sptr.**min_noutput_items**(*unpacked_to_packed_ss_sptr self*) → int

unpacked_to_packed_ss_sptr.**pc_input_buffers_full_avg**(*unpacked_to_packed_ss_sptr self*, *int which*) → float

pc_input_buffers_full_avg(unpacked_to_packed_ss_sptr self) -> pmt_vector_float

unpacked_to_packed_ss_sptr.**pc_noutput_items_avg**(*unpacked_to_packed_ss_sptr self*) → float

unpacked_to_packed_ss_sptr.**pc_nproduced_avg**(*unpacked_to_packed_ss_sptr self*) → float

unpacked_to_packed_ss_sptr.**pc_output_buffers_full_avg**(*unpacked_to_packed_ss_sptr self*, *int which*) → float

pc_output_buffers_full_avg(unpacked_to_packed_ss_sptr self) -> pmt_vector_float

unpacked_to_packed_ss_sptr.**pc_throughput_avg**(*unpacked_to_packed_ss_sptr self*) → float

unpacked_to_packed_ss_sptr.**pc_work_time_avg**(*unpacked_to_packed_ss_sptr self*) → float

unpacked_to_packed_ss_sptr.**pc_work_time_total**(*unpacked_to_packed_ss_sptr self*) → float

unpacked_to_packed_ss_sptr.**sample_delay**(*unpacked_to_packed_ss_sptr self*, *int which*) → unsigned int

unpacked_to_packed_ss_sptr.**set_min_noutput_items**(*unpacked_to_packed_ss_sptr self*, *int m*)

unpacked_to_packed_ss_sptr.**set_thread_priority**(*unpacked_to_packed_ss_sptr self*, *int priority*) → int

unpacked_to_packed_ss_sptr.**thread_priority**(*unpacked_to_packed_ss_sptr self*) → int

gnuradio.blocks.**vco_c**(*double sampling_rate*, *double sensitivity*, *double amplitude*) → vco_c_sptr

VCO - Voltage controlled oscillator.

input: float stream of control voltages; output: complex oscillator output

Constructor Specific Documentation:

VCO - Voltage controlled oscillator.

Parameters:
- **sampling_rate** – sampling rate (Hz)
- **sensitivity** – units are radians/sec/volt
- **amplitude** – output amplitude

vco_c_sptr.**active_thread_priority**(*vco_c_sptr self*) → int

vco_c_sptr.**declare_sample_delay**(*vco_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(vco_c_sptr self, unsigned int delay)

vco_c_sptr.**message_subscribers**(*vco_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vco_c_sptr.**min_noutput_items**(*vco_c_sptr self*) → int

vco_c_sptr.**pc_input_buffers_full_avg**(*vco_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vco_c_sptr self) -> pmt_vector_float

vco_c_sptr.**pc_noutput_items_avg**(*vco_c_sptr self*) → float

vco_c_sptr.**pc_nproduced_avg**(*vco_c_sptr self*) → float

vco_c_sptr.**pc_output_buffers_full_avg**(*vco_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vco_c_sptr self) -> pmt_vector_float

vco_c_sptr.**pc_throughput_avg**(*vco_c_sptr self*) → float

vco_c_sptr.**pc_work_time_avg**(*vco_c_sptr self*) → float

vco_c_sptr.**pc_work_time_total**(*vco_c_sptr self*) → float

vco_c_sptr.**sample_delay**(*vco_c_sptr self*, *int which*) → unsigned int

vco_c_sptr.**set_min_noutput_items**(*vco_c_sptr self*, *int m*)

vco_c_sptr.**set_thread_priority**(*vco_c_sptr self*, *int priority*) → int

vco_c_sptr.**thread_priority**(*vco_c_sptr self*) → int

gnuradio.blocks.**vco_f**(*double sampling_rate*, *double sensitivity*, *double amplitude*) → vco_f_sptr

VCO - Voltage controlled oscillator.

input: float stream of control voltages; output: float oscillator output

Constructor Specific Documentation:

VCO - Voltage controlled oscillator.

Parameters:
- **sampling_rate** – sampling rate (Hz)
- **sensitivity** – units are radians/sec/volt
- **amplitude** – output amplitude

vco_f_sptr.**active_thread_priority**(*vco_f_sptr self*) → int

vco_f_sptr.**declare_sample_delay**(*vco_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(vco_f_sptr self, unsigned int delay)

vco_f_sptr.**message_subscribers**(*vco_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vco_f_sptr.**min_noutput_items**(*vco_f_sptr self*) → int

vco_f_sptr.**pc_input_buffers_full_avg**(*vco_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vco_f_sptr self) -> pmt_vector_float

vco_f_sptr.**pc_noutput_items_avg**(*vco_f_sptr self*) → float

vco_f_sptr.**pc_nproduced_avg**(*vco_f_sptr self*) → float

vco_f_sptr.**pc_output_buffers_full_avg**(*vco_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vco_f_sptr self) -> pmt_vector_float

vco_f_sptr.**pc_throughput_avg**(*vco_f_sptr self*) → float

vco_f_sptr.**pc_work_time_avg**(*vco_f_sptr self*) → float

vco_f_sptr.**pc_work_time_total**(*vco_f_sptr self*) → float

vco_f_sptr.**sample_delay**(*vco_f_sptr self*, *int which*) → unsigned int

vco_f_sptr.**set_min_noutput_items**(*vco_f_sptr self*, *int m*)

vco_f_sptr.**set_thread_priority**(*vco_f_sptr self*, *int priority*) → int

vco_f_sptr.**thread_priority**(*vco_f_sptr self*) → int

gnuradio.blocks.**vector_insert_b**(*std::vector< unsigned char, std::allocator< unsigned char > > const & data*, *int periodicity*, *int offset=0*) → vector_insert_b_sptr

source of unsigned char's that gets its data from a vector

Constructor Specific Documentation:

Make vector insert block.

Parameters:
- **data** – vector of data to insert
- **periodicity** – number of samples between when to send
- **offset** – initial item offset of first insert

vector_insert_b_sptr.**active_thread_priority**(*vector_insert_b_sptr self*) → int

vector_insert_b_sptr.**declare_sample_delay**(*vector_insert_b_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_insert_b_sptr self, unsigned int delay)

vector_insert_b_sptr.**message_subscribers**(*vector_insert_b_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_insert_b_sptr.**min_noutput_items**(*vector_insert_b_sptr self*) → int

vector_insert_b_sptr.**pc_input_buffers_full_avg**(*vector_insert_b_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_insert_b_sptr self) -> pmt_vector_float

vector_insert_b_sptr.**pc_noutput_items_avg**(*vector_insert_b_sptr self*) → float

vector_insert_b_sptr.**pc_nproduced_avg**(*vector_insert_b_sptr self*) → float

vector_insert_b_sptr.**pc_output_buffers_full_avg**(*vector_insert_b_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_insert_b_sptr self) -> pmt_vector_float

vector_insert_b_sptr.**pc_throughput_avg**(*vector_insert_b_sptr self*) → float

vector_insert_b_sptr.**pc_work_time_avg**(*vector_insert_b_sptr self*) → float

vector_insert_b_sptr.**pc_work_time_total**(*vector_insert_b_sptr self*) → float

vector_insert_b_sptr.**rewind**(*vector_insert_b_sptr self*)

vector_insert_b_sptr.**sample_delay**(*vector_insert_b_sptr self*, *int which*) → unsigned int

vector_insert_b_sptr.**set_data**(*vector_insert_b_sptr self*, *std::vector< unsigned char, std::allocator< unsigned char > > const & data*)

vector_insert_b_sptr.**set_min_noutput_items**(*vector_insert_b_sptr self*, *int m*)

vector_insert_b_sptr.**set_thread_priority**(*vector_insert_b_sptr self*, *int priority*) → int

vector_insert_b_sptr.**thread_priority**(*vector_insert_b_sptr self*) → int

gnuradio.blocks.**vector_insert_c**(*pmt_vector_cfloat data*, *int periodicity*, *int offset=0*) → vector_insert_c_sptr

source of gr_complex's that gets its data from a vector

Constructor Specific Documentation:

Make vector insert block.

Parameters:
- **data** – vector of data to insert
- **periodicity** – number of samples between when to send
- **offset** – initial item offset of first insert

vector_insert_c_sptr.**active_thread_priority**(*vector_insert_c_sptr self*) → int

vector_insert_c_sptr.**declare_sample_delay**(*vector_insert_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_insert_c_sptr self, unsigned int delay)

vector_insert_c_sptr.**message_subscribers**(*vector_insert_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_insert_c_sptr.**min_noutput_items**(*vector_insert_c_sptr self*) → int

vector_insert_c_sptr.**pc_input_buffers_full_avg**(*vector_insert_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_insert_c_sptr self) -> pmt_vector_float

vector_insert_c_sptr.**pc_noutput_items_avg**(*vector_insert_c_sptr self*) → float

vector_insert_c_sptr.**pc_nproduced_avg**(*vector_insert_c_sptr self*) → float

vector_insert_c_sptr.**pc_output_buffers_full_avg**(*vector_insert_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_insert_c_sptr self) -> pmt_vector_float

vector_insert_c_sptr.**pc_throughput_avg**(*vector_insert_c_sptr self*) → float

vector_insert_c_sptr.**pc_work_time_avg**(*vector_insert_c_sptr self*) → float

vector_insert_c_sptr.**pc_work_time_total**(*vector_insert_c_sptr self*) → float

vector_insert_c_sptr.**rewind**(*vector_insert_c_sptr self*)

vector_insert_c_sptr.**sample_delay**(*vector_insert_c_sptr self*, *int which*) → unsigned int

vector_insert_c_sptr.**set_data**(*vector_insert_c_sptr self*, *pmt_vector_cfloat data*)

vector_insert_c_sptr.**set_min_noutput_items**(*vector_insert_c_sptr self*, *int m*)

vector_insert_c_sptr.**set_thread_priority**(*vector_insert_c_sptr self*, *int priority*) → int

vector_insert_c_sptr.**thread_priority**(*vector_insert_c_sptr self*) → int

gnuradio.blocks.**vector_insert_f**(*pmt_vector_float data*, *int periodicity*, *int offset=0*) →

vector_insert_f_sptr

source of float's that gets its data from a vector

Constructor Specific Documentation:

Make vector insert block.

Parameters:
- **data** – vector of data to insert
- **periodicity** – number of samples between when to send
- **offset** – initial item offset of first insert

vector_insert_f_sptr.**active_thread_priority**(*vector_insert_f_sptr self*) → int

vector_insert_f_sptr.**declare_sample_delay**(*vector_insert_f_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_insert_f_sptr self, unsigned int delay)

vector_insert_f_sptr.**message_subscribers**(*vector_insert_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_insert_f_sptr.**min_noutput_items**(*vector_insert_f_sptr self*) → int

vector_insert_f_sptr.**pc_input_buffers_full_avg**(*vector_insert_f_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_insert_f_sptr self) -> pmt_vector_float

vector_insert_f_sptr.**pc_noutput_items_avg**(*vector_insert_f_sptr self*) → float

vector_insert_f_sptr.**pc_nproduced_avg**(*vector_insert_f_sptr self*) → float

vector_insert_f_sptr.**pc_output_buffers_full_avg**(*vector_insert_f_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_insert_f_sptr self) -> pmt_vector_float

vector_insert_f_sptr.**pc_throughput_avg**(*vector_insert_f_sptr self*) → float

vector_insert_f_sptr.**pc_work_time_avg**(*vector_insert_f_sptr self*) → float

vector_insert_f_sptr.**pc_work_time_total**(*vector_insert_f_sptr self*) → float

vector_insert_f_sptr.**rewind**(*vector_insert_f_sptr self*)

vector_insert_f_sptr.**sample_delay**(*vector_insert_f_sptr self*, *int which*) → unsigned int

vector_insert_f_sptr.**set_data**(*vector_insert_f_sptr self*, *pmt_vector_float data*)

vector_insert_f_sptr.**set_min_noutput_items**(*vector_insert_f_sptr self*, *int m*)

vector_insert_f_sptr.**set_thread_priority**(*vector_insert_f_sptr self*, *int priority*) → int

vector_insert_f_sptr.**thread_priority**(*vector_insert_f_sptr self*) → int

gnuradio.blocks.**vector_insert_i**(*std::vector< int, std::allocator< int > > const & data*, *int periodicity*, *int offset=0*) → vector_insert_i_sptr

source of int's that gets its data from a vector

Constructor Specific Documentation:

Make vector insert block.

Parameters:
- **data** – vector of data to insert
- **periodicity** – number of samples between when to send
- **offset** – initial item offset of first insert

vector_insert_i_sptr.**active_thread_priority**(*vector_insert_i_sptr self*) → int

vector_insert_i_sptr.**declare_sample_delay**(*vector_insert_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_insert_i_sptr self, unsigned int delay)

vector_insert_i_sptr.**message_subscribers**(*vector_insert_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_insert_i_sptr.**min_noutput_items**(*vector_insert_i_sptr self*) → int

vector_insert_i_sptr.**pc_input_buffers_full_avg**(*vector_insert_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_insert_i_sptr self) -> pmt_vector_float

vector_insert_i_sptr.**pc_noutput_items_avg**(*vector_insert_i_sptr self*) → float

vector_insert_i_sptr.**pc_nproduced_avg**(*vector_insert_i_sptr self*) → float

vector_insert_i_sptr.**pc_output_buffers_full_avg**(*vector_insert_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_insert_i_sptr self) -> pmt_vector_float

vector_insert_i_sptr.**pc_throughput_avg**(*vector_insert_i_sptr self*) → float

vector_insert_i_sptr.**pc_work_time_avg**(*vector_insert_i_sptr self*) → float

vector_insert_i_sptr.**pc_work_time_total**(*vector_insert_i_sptr self*) → float

vector_insert_i_sptr.**rewind**(*vector_insert_i_sptr self*)

vector_insert_i_sptr.**sample_delay**(*vector_insert_i_sptr self*, *int which*) → unsigned int

vector_insert_i_sptr.**set_data**(*vector_insert_i_sptr self*, *std::vector< int, std::allocator< int > > const & data*)

vector_insert_i_sptr.**set_min_noutput_items**(*vector_insert_i_sptr self*, *int m*)

vector_insert_i_sptr.**set_thread_priority**(*vector_insert_i_sptr self*, *int priority*) → int

vector_insert_i_sptr.**thread_priority**(*vector_insert_i_sptr self*) → int

gnuradio.blocks.**vector_insert_s**(*std::vector< short, std::allocator< short > > const & data*, *int periodicity*, *int offset=0*) → vector_insert_s_sptr

source of short's that gets its data from a vector

Constructor Specific Documentation:

Make vector insert block.

Parameters:
- **data** – vector of data to insert
- **periodicity** – number of samples between when to send
- **offset** – initial item offset of first insert

vector_insert_s_sptr.**active_thread_priority**(*vector_insert_s_sptr self*) → int

vector_insert_s_sptr.**declare_sample_delay**(*vector_insert_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_insert_s_sptr self, unsigned int delay)

vector_insert_s_sptr.**message_subscribers**(*vector_insert_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_insert_s_sptr.**min_noutput_items**(*vector_insert_s_sptr self*) → int

vector_insert_s_sptr.**pc_input_buffers_full_avg**(*vector_insert_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_insert_s_sptr self) -> pmt_vector_float

vector_insert_s_sptr.**pc_noutput_items_avg**(*vector_insert_s_sptr self*) → float

vector_insert_s_sptr.**pc_nproduced_avg**(*vector_insert_s_sptr self*) → float

vector_insert_s_sptr.**pc_output_buffers_full_avg**(*vector_insert_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_insert_s_sptr self) -> pmt_vector_float

vector_insert_s_sptr.**pc_throughput_avg**(*vector_insert_s_sptr self*) → float

vector_insert_s_sptr.**pc_work_time_avg**(*vector_insert_s_sptr self*) → float

vector_insert_s_sptr.**pc_work_time_total**(*vector_insert_s_sptr self*) → float

vector_insert_s_sptr.**rewind**(*vector_insert_s_sptr self*)

vector_insert_s_sptr.**sample_delay**(*vector_insert_s_sptr self*, *int which*) → unsigned int

vector_insert_s_sptr.**set_data**(*vector_insert_s_sptr self*, *std::vector< short, std::allocator< short > > const & data*)

vector_insert_s_sptr.**set_min_noutput_items**(*vector_insert_s_sptr self*, *int m*)

vector_insert_s_sptr.**set_thread_priority**(*vector_insert_s_sptr self*, *int priority*) → int

vector_insert_s_sptr.**thread_priority**(*vector_insert_s_sptr self*) → int

gnuradio.blocks.**vector_map**(*size_t item_size*, *gr_vsize_t in_vlens*, *gr_vvvsize_t mapping*) → vector_map_sptr

Maps elements from a set of input vectors to a set of output vectors.

If in[i] is the input vector in the i'th stream then the output vector in the j'th stream is:

out[j][k] = in[mapping[j][k][0]][mapping[j][k][1]]

That is mapping is of the form (out_stream1_mapping, out_stream2_mapping, ...) and out_stream1_mapping is of the form (element1_mapping, element2_mapping, ...) and element1_mapping is of the form (in_stream, in_element).

Constructor Specific Documentation:

Build a vector map block.

Parameters:
- **item_size** – (integer) size of vector elements
- **in_vlens** – (vector of integers) number of elements in each input vector
- **mapping** – (vector of vectors of vectors of integers) how to map elements from input to output vectors

vector_map_sptr.**active_thread_priority**(*vector_map_sptr self*) → int

vector_map_sptr.**declare_sample_delay**(*vector_map_sptr self*, *int which*, *int delay*)
declare_sample_delay(vector_map_sptr self, unsigned int delay)

vector_map_sptr.**message_subscribers**(*vector_map_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_map_sptr.**min_noutput_items**(*vector_map_sptr self*) → int

vector_map_sptr.**pc_input_buffers_full_avg**(*vector_map_sptr self*, *int which*) → float
pc_input_buffers_full_avg(vector_map_sptr self) -> pmt_vector_float

vector_map_sptr.**pc_noutput_items_avg**(*vector_map_sptr self*) → float

vector_map_sptr.**pc_nproduced_avg**(*vector_map_sptr self*) → float

vector_map_sptr.**pc_output_buffers_full_avg**(*vector_map_sptr self*, *int which*) → float
pc_output_buffers_full_avg(vector_map_sptr self) -> pmt_vector_float

vector_map_sptr.**pc_throughput_avg**(*vector_map_sptr self*) → float

vector_map_sptr.**pc_work_time_avg**(*vector_map_sptr self*) → float

vector_map_sptr.**pc_work_time_total**(*vector_map_sptr self*) → float

vector_map_sptr.**sample_delay**(*vector_map_sptr self*, *int which*) → unsigned int

vector_map_sptr.**set_mapping**(*vector_map_sptr self*, *gr_vvvsize_t mapping*)

vector_map_sptr.**set_min_noutput_items**(*vector_map_sptr self*, *int m*)

vector_map_sptr.**set_thread_priority**(*vector_map_sptr self*, *int priority*) → int

vector_map_sptr.**thread_priority**(*vector_map_sptr self*) → int

gnuradio.blocks.**vector_sink_b**(*int const vlen=1*, *int const reserve_items=1024*) → vector_sink_b_sptr
unsigned char sink that writes to a vector

Constructor Specific Documentation:

Make a new instance of the vector source, and return a shared pointer to it.

Parameters:
- **vlen** – length of vector items
- **reserve_items** – reserve space in the internal storage for this many items; the internal storage will still grow to accommodate more item if necessary, but setting this to a realistic value can avoid memory allocations during runtime, especially if you know a priori how many items you're going to store.

vector_sink_b_sptr.**active_thread_priority**(*vector_sink_b_sptr self*) → int

vector_sink_b_sptr.**data**(*vector_sink_b_sptr self*) → std::vector< unsigned char,std::allocator< unsigned char > >

vector_sink_b_sptr.**declare_sample_delay**(*vector_sink_b_sptr self*, *int which*, *int delay*)
declare_sample_delay(vector_sink_b_sptr self, unsigned int delay)

vector_sink_b_sptr.**message_subscribers**(*vector_sink_b_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_sink_b_sptr.**min_noutput_items**(*vector_sink_b_sptr self*) → int

vector_sink_b_sptr.**pc_input_buffers_full_avg**(*vector_sink_b_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_sink_b_sptr self) -> pmt_vector_float

vector_sink_b_sptr.**pc_noutput_items_avg**(*vector_sink_b_sptr self*) → float

vector_sink_b_sptr.**pc_nproduced_avg**(*vector_sink_b_sptr self*) → float

vector_sink_b_sptr.**pc_output_buffers_full_avg**(*vector_sink_b_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_sink_b_sptr self) -> pmt_vector_float

vector_sink_b_sptr.**pc_throughput_avg**(*vector_sink_b_sptr self*) → float

vector_sink_b_sptr.**pc_work_time_avg**(*vector_sink_b_sptr self*) → float

vector_sink_b_sptr.**pc_work_time_total**(*vector_sink_b_sptr self*) → float

vector_sink_b_sptr.**reset**(*vector_sink_b_sptr self*)

Clear the data and tags containers.

vector_sink_b_sptr.**sample_delay**(*vector_sink_b_sptr self*, *int which*) → unsigned int

vector_sink_b_sptr.**set_min_noutput_items**(*vector_sink_b_sptr self*, *int m*)

vector_sink_b_sptr.**set_thread_priority**(*vector_sink_b_sptr self*, *int priority*) → int

vector_sink_b_sptr.**tags**(*vector_sink_b_sptr self*) → tags_vector_t

vector_sink_b_sptr.**thread_priority**(*vector_sink_b_sptr self*) → int

gnuradio.blocks.**vector_sink_c**(*int const vlen=1*, *int const reserve_items=1024*) → vector_sink_c_sptr

gr_complex sink that writes to a vector

Constructor Specific Documentation:

Make a new instance of the vector source, and return a shared pointer to it.

Parameters:
- **vlen** – length of vector items
- **reserve_items** – reserve space in the internal storage for this many items; the internal storage will still grow to accommodate more item if necessary, but setting this to a realistic value can avoid memory allocations during runtime, especially if you know a priori how many items you're going to store.

vector_sink_c_sptr.**active_thread_priority**(*vector_sink_c_sptr self*) → int

vector_sink_c_sptr.**data**(*vector_sink_c_sptr self*) → pmt_vector_cfloat

vector_sink_c_sptr.**declare_sample_delay**(*vector_sink_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_sink_c_sptr self, unsigned int delay)

vector_sink_c_sptr.**message_subscribers**(*vector_sink_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_sink_c_sptr.**min_noutput_items**(*vector_sink_c_sptr self*) → int

vector_sink_c_sptr.**pc_input_buffers_full_avg**(*vector_sink_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_sink_c_sptr self) -> pmt_vector_float

vector_sink_c_sptr.**pc_noutput_items_avg**(*vector_sink_c_sptr self*) → float

vector_sink_c_sptr.**pc_nproduced_avg**(*vector_sink_c_sptr self*) → float

vector_sink_c_sptr.**pc_output_buffers_full_avg**(*vector_sink_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_sink_c_sptr self) -> pmt_vector_float

vector_sink_c_sptr.**pc_throughput_avg**(*vector_sink_c_sptr self*) → float

vector_sink_c_sptr.**pc_work_time_avg**(*vector_sink_c_sptr self*) → float

vector_sink_c_sptr.**pc_work_time_total**(*vector_sink_c_sptr self*) → float

vector_sink_c_sptr.**reset**(*vector_sink_c_sptr self*)

Clear the data and tags containers.

vector_sink_c_sptr.**sample_delay**(*vector_sink_c_sptr self*, *int which*) → unsigned int

vector_sink_c_sptr.**set_min_noutput_items**(*vector_sink_c_sptr self*, *int m*)

vector_sink_c_sptr.**set_thread_priority**(*vector_sink_c_sptr self*, *int priority*) → int

vector_sink_c_sptr.**tags**(*vector_sink_c_sptr self*) → tags_vector_t

vector_sink_c_sptr.**thread_priority**(*vector_sink_c_sptr self*) → int

gnuradio.blocks.**vector_sink_f**(*int const vlen=1*, *int const reserve_items=1024*) → vector_sink_f_sptr

float sink that writes to a vector

Constructor Specific Documentation:

Make a new instance of the vector source, and return a shared pointer to it.

Parameters:
- **vlen** – length of vector items
- **reserve_items** – reserve space in the internal storage for this many items; the internal storage will still grow to accommodate more item if necessary, but setting this to a realistic value can avoid memory allocations during runtime, especially if you know a priori how many items you're going to store.

vector_sink_f_sptr.**active_thread_priority**(*vector_sink_f_sptr self*) → int

vector_sink_f_sptr.**data**(*vector_sink_f_sptr self*) → pmt_vector_float

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.**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.**reset**(*vector_sink_f_sptr self*)

Clear the data and tags containers.

vector_sink_f_sptr.**sample_delay**(*vector_sink_f_sptr self*, *int which*) → unsigned int

vector_sink_f_sptr.**set_min_noutput_items**(*vector_sink_f_sptr self*, *int m*)

vector_sink_f_sptr.**set_thread_priority**(*vector_sink_f_sptr self*, *int priority*) → int

vector_sink_f_sptr.**tags**(*vector_sink_f_sptr self*) → tags_vector_t

vector_sink_f_sptr.**thread_priority**(*vector_sink_f_sptr self*) → int

gnuradio.blocks.**vector_sink_i**(*int const vlen=1*, *int const reserve_items=1024*) → vector_sink_i_sptr

int sink that writes to a vector

Constructor Specific Documentation:

Make a new instance of the vector source, and return a shared pointer to it.

Parameters:
- **vlen** – length of vector items
- **reserve_items** – reserve space in the internal storage for this many items; the internal storage will still grow to accommodate more item if necessary, but setting this to a realistic value can avoid memory allocations during runtime, especially if you know a priori how many items you're going to store.

vector_sink_i_sptr.**active_thread_priority**(*vector_sink_i_sptr self*) → int

vector_sink_i_sptr.**data**(*vector_sink_i_sptr self*) → std::vector< int,std::allocator< int > >

vector_sink_i_sptr.**declare_sample_delay**(*vector_sink_i_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_sink_i_sptr self, unsigned int delay)

vector_sink_i_sptr.**message_subscribers**(*vector_sink_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_sink_i_sptr.**min_noutput_items**(*vector_sink_i_sptr self*) → int

vector_sink_i_sptr.**pc_input_buffers_full_avg**(*vector_sink_i_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_sink_i_sptr self) -> pmt_vector_float

vector_sink_i_sptr.**pc_noutput_items_avg**(*vector_sink_i_sptr self*) → float

vector_sink_i_sptr.**pc_nproduced_avg**(*vector_sink_i_sptr self*) → float

vector_sink_i_sptr.**pc_output_buffers_full_avg**(*vector_sink_i_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_sink_i_sptr self) -> pmt_vector_float

vector_sink_i_sptr.**pc_throughput_avg**(*vector_sink_i_sptr self*) → float

vector_sink_i_sptr.**pc_work_time_avg**(*vector_sink_i_sptr self*) → float

vector_sink_i_sptr.**pc_work_time_total**(*vector_sink_i_sptr self*) → float

vector_sink_i_sptr.**reset**(*vector_sink_i_sptr self*)

Clear the data and tags containers.

vector_sink_i_sptr.**sample_delay**(*vector_sink_i_sptr self*, *int which*) → unsigned int

vector_sink_i_sptr.**set_min_noutput_items**(*vector_sink_i_sptr self*, *int m*)

vector_sink_i_sptr.**set_thread_priority**(*vector_sink_i_sptr self*, *int priority*) → int

vector_sink_i_sptr.**tags**(*vector_sink_i_sptr self*) → tags_vector_t

vector_sink_i_sptr.**thread_priority**(*vector_sink_i_sptr self*) → int

gnuradio.blocks.**vector_sink_s**(*int const vlen=1*, *int const reserve_items=1024*) → vector_sink_s_sptr

short sink that writes to a vector

Constructor Specific Documentation:

Make a new instance of the vector source, and return a shared pointer to it.

Parameters:

- **vlen** – length of vector items
- **reserve_items** – reserve space in the internal storage for this many items; the internal storage will still grow to accommodate more item if necessary, but setting this to a realistic value can avoid memory allocations during runtime, especially if you know a priori how many items you're going to store.

vector_sink_s_sptr.**active_thread_priority**(*vector_sink_s_sptr self*) → int

vector_sink_s_sptr.**data**(*vector_sink_s_sptr self*) → std::vector< short,std::allocator< short > >

vector_sink_s_sptr.**declare_sample_delay**(*vector_sink_s_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_sink_s_sptr self, unsigned int delay)

vector_sink_s_sptr.**message_subscribers**(*vector_sink_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_sink_s_sptr.**min_noutput_items**(*vector_sink_s_sptr self*) → int

vector_sink_s_sptr.**pc_input_buffers_full_avg**(*vector_sink_s_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_sink_s_sptr self) -> pmt_vector_float

vector_sink_s_sptr.**pc_noutput_items_avg**(*vector_sink_s_sptr self*) → float

vector_sink_s_sptr.**pc_nproduced_avg**(*vector_sink_s_sptr self*) → float

vector_sink_s_sptr.**pc_output_buffers_full_avg**(*vector_sink_s_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_sink_s_sptr self) -> pmt_vector_float

vector_sink_s_sptr.**pc_throughput_avg**(*vector_sink_s_sptr self*) → float

vector_sink_s_sptr.**pc_work_time_avg**(*vector_sink_s_sptr self*) → float

vector_sink_s_sptr.**pc_work_time_total**(*vector_sink_s_sptr self*) → float

vector_sink_s_sptr.**reset**(*vector_sink_s_sptr self*)

Clear the data and tags containers.

vector_sink_s_sptr.**sample_delay**(*vector_sink_s_sptr self*, *int which*) → unsigned int

vector_sink_s_sptr.**set_min_noutput_items**(*vector_sink_s_sptr self*, *int m*)

vector_sink_s_sptr.**set_thread_priority**(*vector_sink_s_sptr self*, *int priority*) → int

vector_sink_s_sptr.**tags**(*vector_sink_s_sptr self*) → tags_vector_t

vector_sink_s_sptr.**thread_priority**(*vector_sink_s_sptr self*) → int

gnuradio.blocks.**vector_source_b**(*std::vector< unsigned char, std::allocator< unsigned char > > const & data*, *bool repeat=False*, *int vlen=1*, *tags_vector_t tags*) → vector_source_b_sptr

Source that streams unsigned char items based on the input vector.

This block produces a stream of samples based on an input vector. In C++, this is a std::vector<unsigned char>, and in Python, this is either a list or tuple. The data can repeat infinitely until the flowgraph is terminated by some other event or, the default, run the data once and stop.

The vector source can also produce stream tags with the data. Pass in a vector of gr::tag_t objects and they will be emitted based on the specified offset of the tag.

GNU Radio provides a utility Python module in gr.tag_utils to convert between tags and Python objects: gr.tag_utils.python_to_tag.

We can create tags as Python lists (or tuples) using the list structure [int offset, pmt key, pmt value, pmt srcid]. It is important to define the list/tuple with the values in the correct order and with the correct data type. A python dictionary can also be used using the keys: “offset”, “key”, “value”, and “srcid” with the same data types as for the lists.

When given a list of tags, the vector source will emit the tags repeatedly by updating the offset relative to the vector stream length. That is, if the vector has 500 items and a tag has an offset of 0, that tag will be placed on item 0, 500, 1000, 1500, etc.

Constructor Specific Documentation:

Parameters:
- **data** –
- **repeat** –
- **vlen** –
- **tags** –

vector_source_b_sptr.**active_thread_priority**(*vector_source_b_sptr self*) → int

vector_source_b_sptr.**declare_sample_delay**(*vector_source_b_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_source_b_sptr self, unsigned int delay)

vector_source_b_sptr.**message_subscribers**(*vector_source_b_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_source_b_sptr.**min_noutput_items**(*vector_source_b_sptr self*) → int

vector_source_b_sptr.**pc_input_buffers_full_avg**(*vector_source_b_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_source_b_sptr self) -> pmt_vector_float

vector_source_b_sptr.**pc_noutput_items_avg**(*vector_source_b_sptr self*) → float

vector_source_b_sptr.**pc_nproduced_avg**(*vector_source_b_sptr self*) → float

vector_source_b_sptr.**pc_output_buffers_full_avg**(*vector_source_b_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_source_b_sptr self) -> pmt_vector_float

vector_source_b_sptr.**pc_throughput_avg**(*vector_source_b_sptr self*) → float

vector_source_b_sptr.**pc_work_time_avg**(*vector_source_b_sptr self*) → float

vector_source_b_sptr.**pc_work_time_total**(*vector_source_b_sptr self*) → float

vector_source_b_sptr.**rewind**(*vector_source_b_sptr self*)

vector_source_b_sptr.**sample_delay**(*vector_source_b_sptr self*, *int which*) → unsigned int

vector_source_b_sptr.**set_data**(*vector_source_b_sptr self*, *std::vector< unsigned char, std::allocator< unsigned char > > const & data*, *tags_vector_t tags*)

vector_source_b_sptr.**set_min_noutput_items**(*vector_source_b_sptr self*, *int m*)

vector_source_b_sptr.**set_repeat**(*vector_source_b_sptr self*, *bool repeat*)

vector_source_b_sptr.**set_thread_priority**(*vector_source_b_sptr self*, *int priority*) → int

vector_source_b_sptr.**thread_priority**(*vector_source_b_sptr self*) → int

gnuradio.blocks.**vector_source_c**(*pmt_vector_cfloat data*, *bool repeat=False*, *int vlen=1*, *tags_vector_t tags*) → vector_source_c_sptr

Source that streams gr_complex items based on the input vector.

This block produces a stream of samples based on an input vector. In C++, this is a std::vector<gr_complex>, and in Python, this is either a list or tuple. The data can repeat infinitely until the flowgraph is terminated by some other event or, the default, run the data once and stop.

The vector source can also produce stream tags with the data. Pass in a vector of gr::tag_t objects and they will be emitted based on the specified offset of the tag.

GNU Radio provides a utility Python module in gr.tag_utils to convert between tags and Python objects: gr.tag_utils.python_to_tag.

We can create tags as Python lists (or tuples) using the list structure [int offset, pmt key, pmt value, pmt srcid]. It is important to define the list/tuple with the values in the correct order and with the correct data type. A python dictionary can also be used using the keys: “offset”, “key”, “value”, and “srcid” with the same data types as for the lists.

When given a list of tags, the vector source will emit the tags repeatedly by updating the offset relative to the vector stream length. That is, if the vector has 500 items and a tag has an offset of 0, that tag will be placed on item 0, 500, 1000, 1500, etc.

Constructor Specific Documentation:

Parameters:
- **data** –
- **repeat** –
- **vlen** –
- **tags** –

vector_source_c_sptr.**active_thread_priority**(*vector_source_c_sptr self*) → int

vector_source_c_sptr.**declare_sample_delay**(*vector_source_c_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_source_c_sptr self, unsigned int delay)

vector_source_c_sptr.**message_subscribers**(*vector_source_c_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_source_c_sptr.**min_noutput_items**(*vector_source_c_sptr self*) → int

vector_source_c_sptr.**pc_input_buffers_full_avg**(*vector_source_c_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_source_c_sptr self) -> pmt_vector_float

vector_source_c_sptr.**pc_noutput_items_avg**(*vector_source_c_sptr self*) → float

vector_source_c_sptr.**pc_nproduced_avg**(*vector_source_c_sptr self*) → float

vector_source_c_sptr.**pc_output_buffers_full_avg**(*vector_source_c_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_source_c_sptr self) -> pmt_vector_float

vector_source_c_sptr.**pc_throughput_avg**(*vector_source_c_sptr self*) → float

vector_source_c_sptr.**pc_work_time_avg**(*vector_source_c_sptr self*) → float

vector_source_c_sptr.**pc_work_time_total**(*vector_source_c_sptr self*) → float

vector_source_c_sptr.**rewind**(*vector_source_c_sptr self*)

vector_source_c_sptr.**sample_delay**(*vector_source_c_sptr self*, *int which*) → unsigned int

vector_source_c_sptr.**set_data**(*vector_source_c_sptr self*, *pmt_vector_cfloat data*, *tags_vector_t tags*)

vector_source_c_sptr.**set_min_noutput_items**(*vector_source_c_sptr self*, *int m*)

vector_source_c_sptr.**set_repeat**(*vector_source_c_sptr self*, *bool repeat*)

vector_source_c_sptr.**set_thread_priority**(*vector_source_c_sptr self*, *int priority*) → int

vector_source_c_sptr.**thread_priority**(*vector_source_c_sptr self*) → int

gnuradio.blocks.**vector_source_f**(*pmt_vector_float data*, *bool repeat=False*, *int vlen=1*, *tags_vector_t tags*) → vector_source_f_sptr

Source that streams float items based on the input vector.

This block produces a stream of samples based on an input vector. In C++, this is a std::vector<float>, and in Python, this is either a list or tuple. The data can repeat infinitely until the flowgraph is terminated by some other event or, the default, run the data once and stop.

The vector source can also produce stream tags with the data. Pass in a vector of gr::tag_t objects and they will be emitted based on the specified offset of the tag.

GNU Radio provides a utility Python module in gr.tag_utils to convert between tags and Python objects: gr.tag_utils.python_to_tag.

We can create tags as Python lists (or tuples) using the list structure [int offset, pmt key, pmt value, pmt srcid]. It is important to define the list/tuple with the values in the correct order and with the correct data type. A python dictionary can also be used using the keys: “offset”, “key”, “value”, and “srcid” with the same data types as for the lists.

When given a list of tags, the vector source will emit the tags repeatedly by updating the offset relative to the vector stream length. That is, if the vector has 500 items and a tag has an offset of 0, that tag will be placed on item 0, 500, 1000, 1500, etc.

Constructor Specific Documentation:

Parameters:
- **data** –
- **repeat** –
- **vlen** –
- **tags** –

vector_source_f_sptr.**active_thread_priority**(*vector_source_f_sptr self*) → int

vector_source_f_sptr.**declare_sample_delay**(*vector_source_f_sptr self*, *int which*, *int delay*)
declare_sample_delay(vector_source_f_sptr self, unsigned int delay)

vector_source_f_sptr.**message_subscribers**(*vector_source_f_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_source_f_sptr.**min_noutput_items**(*vector_source_f_sptr self*) → int

vector_source_f_sptr.**pc_input_buffers_full_avg**(*vector_source_f_sptr self*, *int which*) → float
pc_input_buffers_full_avg(vector_source_f_sptr self) -> pmt_vector_float

vector_source_f_sptr.**pc_noutput_items_avg**(*vector_source_f_sptr self*) → float

vector_source_f_sptr.**pc_nproduced_avg**(*vector_source_f_sptr self*) → float

vector_source_f_sptr.**pc_output_buffers_full_avg**(*vector_source_f_sptr self*, *int which*) → float
pc_output_buffers_full_avg(vector_source_f_sptr self) -> pmt_vector_float

vector_source_f_sptr.**pc_throughput_avg**(*vector_source_f_sptr self*) → float

vector_source_f_sptr.**pc_work_time_avg**(*vector_source_f_sptr self*) → float

vector_source_f_sptr.**pc_work_time_total**(*vector_source_f_sptr self*) → float

vector_source_f_sptr.**rewind**(*vector_source_f_sptr self*)

vector_source_f_sptr.**sample_delay**(*vector_source_f_sptr self*, *int which*) → unsigned int

vector_source_f_sptr.**set_data**(*vector_source_f_sptr self*, *pmt_vector_float data*, *tags_vector_t tags*)

vector_source_f_sptr.**set_min_noutput_items**(*vector_source_f_sptr self*, *int m*)

vector_source_f_sptr.**set_repeat**(*vector_source_f_sptr self*, *bool repeat*)

vector_source_f_sptr.**set_thread_priority**(*vector_source_f_sptr self*, *int priority*) → int

vector_source_f_sptr.**thread_priority**(*vector_source_f_sptr self*) → int

gnuradio.blocks.**vector_source_i**(*std::vector< int, std::allocator< int > > const & data*, *bool repeat=False*, *int vlen=1*, *tags_vector_t tags*) → vector_source_i_sptr
Source that streams int items based on the input vector.

This block produces a stream of samples based on an input vector. In C++, this is a std::vector<int>, and in Python, this is either a list or tuple. The data can repeat infinitely until the flowgraph is terminated by some other event or, the default, run the data once and stop.

The vector source can also produce stream tags with the data. Pass in a vector of gr::tag_t objects and they will be emitted based on the specified offset of the tag.

GNU Radio provides a utility Python module in gr.tag_utils to convert between tags and Python objects: gr.tag_utils.python_to_tag.

We can create tags as Python lists (or tuples) using the list structure [int offset, pmt key, pmt value, pmt srcid]. It is important to define the list/tuple with the values in the correct order and with the correct data type. A python dictionary can also be used using the keys: “offset”, “key”, “value”, and “srcid” with the same data types as for the lists.

When given a list of tags, the vector source will emit the tags repeatedly by updating the offset relative to the vector stream length. That is, if the vector has 500 items and a tag has an offset of 0, that tag will be placed on item 0, 500, 1000, 1500, etc.

Constructor Specific Documentation:

Parameters:
- **data** –
- **repeat** –
- **vlen** –
- **tags** –

vector_source_i_sptr.**active_thread_priority**(*vector_source_i_sptr self*) → int

vector_source_i_sptr.**declare_sample_delay**(*vector_source_i_sptr self*, *int which*, *int delay*)
declare_sample_delay(vector_source_i_sptr self, unsigned int delay)

vector_source_i_sptr.**message_subscribers**(*vector_source_i_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_source_i_sptr.**min_noutput_items**(*vector_source_i_sptr self*) → int

vector_source_i_sptr.**pc_input_buffers_full_avg**(*vector_source_i_sptr self*, *int which*) → float
pc_input_buffers_full_avg(vector_source_i_sptr self) -> pmt_vector_float

vector_source_i_sptr.**pc_noutput_items_avg**(*vector_source_i_sptr self*) → float

vector_source_i_sptr.**pc_nproduced_avg**(*vector_source_i_sptr self*) → float

vector_source_i_sptr.**pc_output_buffers_full_avg**(*vector_source_i_sptr self*, *int which*) → float
pc_output_buffers_full_avg(vector_source_i_sptr self) -> pmt_vector_float

vector_source_i_sptr.**pc_throughput_avg**(*vector_source_i_sptr self*) → float

vector_source_i_sptr.**pc_work_time_avg**(*vector_source_i_sptr self*) → float

vector_source_i_sptr.**pc_work_time_total**(*vector_source_i_sptr self*) → float

vector_source_i_sptr.**rewind**(*vector_source_i_sptr self*)

vector_source_i_sptr.**sample_delay**(*vector_source_i_sptr self*, *int which*) → unsigned int

vector_source_i_sptr.**set_data**(*vector_source_i_sptr self*, *std::vector< int, std::allocator< int > > const & data*, *tags_vector_t tags*)

vector_source_i_sptr.**set_min_noutput_items**(*vector_source_i_sptr self*, *int m*)

vector_source_i_sptr.**set_repeat**(*vector_source_i_sptr self*, *bool repeat*)

vector_source_i_sptr.**set_thread_priority**(*vector_source_i_sptr self*, *int priority*) → int

vector_source_i_sptr.**thread_priority**(*vector_source_i_sptr self*) → int

gnuradio.blocks.**vector_source_s**(*std::vector< short, std::allocator< short > > const & data*, *bool repeat=False*, *int vlen=1*, *tags_vector_t tags*) → vector_source_s_sptr
Source that streams short items based on the input vector.

This block produces a stream of samples based on an input vector. In C++, this is a std::vector<short>, and in Python, this is either a list or tuple. The data can repeat infinitely until the flowgraph is terminated by some other event or, the default, run the data once and stop.

The vector source can also produce stream tags with the data. Pass in a vector of gr::tag_t objects and they will be emitted based on the specified offset of the tag.

GNU Radio provides a utility Python module in gr.tag_utils to convert between tags and Python objects: gr.tag_utils.python_to_tag.

We can create tags as Python lists (or tuples) using the list structure [int offset, pmt key, pmt value, pmt srcid]. It is important to define the list/tuple with the values in the correct order and with the correct data type. A python dictionary can also be used using the keys: “offset”, “key”, “value”, and “srcid” with the same data types as for the lists.

When given a list of tags, the vector source will emit the tags repeatedly by updating the offset relative to the vector stream length. That is, if the vector has 500 items and a tag has an offset of 0, that tag will be placed on item 0, 500, 1000, 1500, etc.

Constructor Specific Documentation:

Parameters:
- **data** –
- **repeat** –
- **vlen** –
- **tags** –

vector_source_s_sptr.**active_thread_priority**(*vector_source_s_sptr self*) → int

vector_source_s_sptr.**declare_sample_delay**(*vector_source_s_sptr self*, *int which*, *int delay*)
declare_sample_delay(vector_source_s_sptr self, unsigned int delay)

vector_source_s_sptr.**message_subscribers**(*vector_source_s_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_source_s_sptr.**min_noutput_items**(*vector_source_s_sptr self*) → int

vector_source_s_sptr.**pc_input_buffers_full_avg**(*vector_source_s_sptr self*, *int which*) → float
pc_input_buffers_full_avg(vector_source_s_sptr self) -> pmt_vector_float

vector_source_s_sptr.**pc_noutput_items_avg**(*vector_source_s_sptr self*) → float

vector_source_s_sptr.**pc_nproduced_avg**(*vector_source_s_sptr self*) → float

vector_source_s_sptr.**pc_output_buffers_full_avg**(*vector_source_s_sptr self*, *int which*) → float
pc_output_buffers_full_avg(vector_source_s_sptr self) -> pmt_vector_float

vector_source_s_sptr.**pc_throughput_avg**(*vector_source_s_sptr self*) → float

vector_source_s_sptr.**pc_work_time_avg**(*vector_source_s_sptr self*) → float

vector_source_s_sptr.**pc_work_time_total**(*vector_source_s_sptr self*) → float

vector_source_s_sptr.**rewind**(*vector_source_s_sptr self*)

vector_source_s_sptr.**sample_delay**(*vector_source_s_sptr self*, *int which*) → unsigned int

vector_source_s_sptr.**set_data**(*vector_source_s_sptr self*, *std::vector< short, std::allocator< short > > const & data*, *tags_vector_t tags*)

vector_source_s_sptr.**set_min_noutput_items**(*vector_source_s_sptr self*, *int m*)

vector_source_s_sptr.**set_repeat**(*vector_source_s_sptr self*, *bool repeat*)

vector_source_s_sptr.**set_thread_priority**(*vector_source_s_sptr self*, *int priority*) → int

vector_source_s_sptr.**thread_priority**(*vector_source_s_sptr self*) → int

gnuradio.blocks.**vector_to_stream**(*size_t itemsize*, *size_t nitems_per_block*) → vector_to_stream_sptr
convert a stream of gnuradio/blocks of nitems_per_block items into a stream of items

Constructor Specific Documentation:

Make vector-to-stream block

Parameters:
- **itemsize** – the item size of the stream
- **nitems_per_block** – number of items per vector (vector size)

vector_to_stream_sptr.**active_thread_priority**(*vector_to_stream_sptr self*) → int

vector_to_stream_sptr.**declare_sample_delay**(*vector_to_stream_sptr self*, *int which*, *int delay*)
declare_sample_delay(vector_to_stream_sptr self, unsigned int delay)

vector_to_stream_sptr.**message_subscribers**(*vector_to_stream_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_to_stream_sptr.**min_noutput_items**(*vector_to_stream_sptr self*) → int

vector_to_stream_sptr.**pc_input_buffers_full_avg**(*vector_to_stream_sptr self*, *int which*) → float
pc_input_buffers_full_avg(vector_to_stream_sptr self) -> pmt_vector_float

vector_to_stream_sptr.**pc_noutput_items_avg**(*vector_to_stream_sptr self*) → float

vector_to_stream_sptr.**pc_nproduced_avg**(*vector_to_stream_sptr self*) → float

vector_to_stream_sptr.**pc_output_buffers_full_avg**(*vector_to_stream_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_to_stream_sptr self) -> pmt_vector_float

vector_to_stream_sptr.**pc_throughput_avg**(*vector_to_stream_sptr self*) → float

vector_to_stream_sptr.**pc_work_time_avg**(*vector_to_stream_sptr self*) → float

vector_to_stream_sptr.**pc_work_time_total**(*vector_to_stream_sptr self*) → float

vector_to_stream_sptr.**sample_delay**(*vector_to_stream_sptr self*, *int which*) → unsigned int

vector_to_stream_sptr.**set_min_noutput_items**(*vector_to_stream_sptr self*, *int m*)

vector_to_stream_sptr.**set_thread_priority**(*vector_to_stream_sptr self*, *int priority*) → int

vector_to_stream_sptr.**thread_priority**(*vector_to_stream_sptr self*) → int

gnuradio.blocks.**vector_to_streams**(*size_t itemsize*, *size_t nstreams*) → vector_to_streams_sptr

Convert 1 stream of vectors of length N to N streams of items.

Constructor Specific Documentation:

Make vector-to-streams block

Parameters:
- **itemsize** – the item size of the stream
- **nstreams** – number of items per vector (vector size and number of streams produced)

vector_to_streams_sptr.**active_thread_priority**(*vector_to_streams_sptr self*) → int

vector_to_streams_sptr.**declare_sample_delay**(*vector_to_streams_sptr self*, *int which*, *int delay*)

declare_sample_delay(vector_to_streams_sptr self, unsigned int delay)

vector_to_streams_sptr.**message_subscribers**(*vector_to_streams_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

vector_to_streams_sptr.**min_noutput_items**(*vector_to_streams_sptr self*) → int

vector_to_streams_sptr.**pc_input_buffers_full_avg**(*vector_to_streams_sptr self*, *int which*) → float

pc_input_buffers_full_avg(vector_to_streams_sptr self) -> pmt_vector_float

vector_to_streams_sptr.**pc_noutput_items_avg**(*vector_to_streams_sptr self*) → float

vector_to_streams_sptr.**pc_nproduced_avg**(*vector_to_streams_sptr self*) → float

vector_to_streams_sptr.**pc_output_buffers_full_avg**(*vector_to_streams_sptr self*, *int which*) → float

pc_output_buffers_full_avg(vector_to_streams_sptr self) -> pmt_vector_float

vector_to_streams_sptr.**pc_throughput_avg**(*vector_to_streams_sptr self*) → float

vector_to_streams_sptr.**pc_work_time_avg**(*vector_to_streams_sptr self*) → float

vector_to_streams_sptr.**pc_work_time_total**(*vector_to_streams_sptr self*) → float

vector_to_streams_sptr.**sample_delay**(*vector_to_streams_sptr self*, *int which*) → unsigned int

vector_to_streams_sptr.**set_min_noutput_items**(*vector_to_streams_sptr self*, *int m*)

vector_to_streams_sptr.**set_thread_priority**(*vector_to_streams_sptr self*, *int priority*) → int

vector_to_streams_sptr.**thread_priority**(*vector_to_streams_sptr self*) → int

gnuradio.blocks.**wavfile_sink**(*char const * filename*, *int n_channels*, *unsigned int sample_rate*, *int bits_per_sample=16*) → wavfile_sink_sptr

Write stream to a Microsoft PCM (.wav) file.

Values must be floats within [-1;1]. Check gr_make_wavfile_sink() for extra info.

Constructor Specific Documentation:

Parameters:
- **filename** –
- **n_channels** –
- **sample_rate** –
- **bits_per_sample** –

wavfile_sink_sptr.**active_thread_priority**(*wavfile_sink_sptr self*) → int

wavfile_sink_sptr.**close**(*wavfile_sink_sptr self*)

Closes the currently active file and completes the WAV header. Thread-safe.

wavfile_sink_sptr.**declare_sample_delay**(*wavfile_sink_sptr self*, *int which*, *int delay*)

declare_sample_delay(wavfile_sink_sptr self, unsigned int delay)

wavfile_sink_sptr.**message_subscribers**(*wavfile_sink_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

wavfile_sink_sptr.**min_noutput_items**(*wavfile_sink_sptr self*) → int

wavfile_sink_sptr.**open**(*wavfile_sink_sptr self*, *char const * filename*) → bool

Opens a new file and writes a WAV header. Thread-safe.

wavfile_sink_sptr.**pc_input_buffers_full_avg**(*wavfile_sink_sptr self*, *int which*) → float

pc_input_buffers_full_avg(wavfile_sink_sptr self) -> pmt_vector_float

wavfile_sink_sptr.**pc_noutput_items_avg**(*wavfile_sink_sptr self*) → float

wavfile_sink_sptr.**pc_nproduced_avg**(*wavfile_sink_sptr self*) → float

wavfile_sink_sptr.**pc_output_buffers_full_avg**(*wavfile_sink_sptr self*, *int which*) → float

pc_output_buffers_full_avg(wavfile_sink_sptr self) -> pmt_vector_float

wavfile_sink_sptr.**pc_throughput_avg**(*wavfile_sink_sptr self*) → float

wavfile_sink_sptr.**pc_work_time_avg**(*wavfile_sink_sptr self*) → float

wavfile_sink_sptr.**pc_work_time_total**(*wavfile_sink_sptr self*) → float

wavfile_sink_sptr.**sample_delay**(*wavfile_sink_sptr self*, *int which*) → unsigned int

wavfile_sink_sptr.**set_bits_per_sample**(*wavfile_sink_sptr self*, *int bits_per_sample*)

Set bits per sample. This will not affect the WAV file currently opened (see set_sample_rate()). If the value is neither 8 nor 16, the call is ignored and the current value is kept.

wavfile_sink_sptr.**set_min_noutput_items**(*wavfile_sink_sptr self*, *int m*)

wavfile_sink_sptr.**set_sample_rate**(*wavfile_sink_sptr self*, *unsigned int sample_rate*)

Set the sample rate. This will not affect the WAV file currently opened. Any following open() calls will use this new sample rate.

wavfile_sink_sptr.**set_thread_priority**(*wavfile_sink_sptr self*, *int priority*) → int

wavfile_sink_sptr.**thread_priority**(*wavfile_sink_sptr self*) → int

gnuradio.blocks.**wavfile_source**(*char const * filename*, *bool repeat=False*) → wavfile_source_sptr

Read stream from a Microsoft PCM (.wav) file, output floats.

Unless otherwise called, values are within [-1;1]. Check gr_make_wavfile_source() for extra info.

Constructor Specific Documentation:

Parameters:
- **filename** –
- **repeat** –

wavfile_source_sptr.**active_thread_priority**(*wavfile_source_sptr self*) → int

wavfile_source_sptr.**bits_per_sample**(*wavfile_source_sptr self*) → int

Return the number of bits per sample as specified in the wav file header. Only 8 or 16 bit are supported here.

wavfile_source_sptr.**channels**(*wavfile_source_sptr self*) → int

Return the number of channels in the wav file as specified in the wav file header. This is also the max number of outputs you can have.

wavfile_source_sptr.**declare_sample_delay**(*wavfile_source_sptr self*, *int which*, *int delay*)

declare_sample_delay(wavfile_source_sptr self, unsigned int delay)

wavfile_source_sptr.**message_subscribers**(*wavfile_source_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

wavfile_source_sptr.**min_noutput_items**(*wavfile_source_sptr self*) → int

wavfile_source_sptr.**pc_input_buffers_full_avg**(*wavfile_source_sptr self*, *int which*) → float

pc_input_buffers_full_avg(wavfile_source_sptr self) -> pmt_vector_float

wavfile_source_sptr.**pc_noutput_items_avg**(*wavfile_source_sptr self*) → float

wavfile_source_sptr.**pc_nproduced_avg**(*wavfile_source_sptr self*) → float

wavfile_source_sptr.**pc_output_buffers_full_avg**(*wavfile_source_sptr self*, *int which*) → float

pc_output_buffers_full_avg(wavfile_source_sptr self) -> pmt_vector_float

wavfile_source_sptr.**pc_throughput_avg**(*wavfile_source_sptr self*) → float

wavfile_source_sptr.**pc_work_time_avg**(*wavfile_source_sptr self*) → float

wavfile_source_sptr.**pc_work_time_total**(*wavfile_source_sptr self*) → float

wavfile_source_sptr.**sample_delay**(*wavfile_source_sptr self*, *int which*) → unsigned int

wavfile_source_sptr.**sample_rate**(*wavfile_source_sptr self*) → unsigned int

Read the sample rate as specified in the wav file header.

wavfile_source_sptr.**set_min_noutput_items**(*wavfile_source_sptr self*, *int m*)

wavfile_source_sptr.**set_thread_priority**(*wavfile_source_sptr self*, *int priority*) → int

wavfile_source_sptr.**thread_priority**(*wavfile_source_sptr self*) → int

gnuradio.blocks.**xor_bb**(*size_t vlen=1*) → xor_bb_sptr

output = input_0 ^ input_1 ^ , ... ^ input_N)

Bitwise boolean xor across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

xor_bb_sptr.**active_thread_priority**(*xor_bb_sptr self*) → int

xor_bb_sptr.**declare_sample_delay**(*xor_bb_sptr self*, *int which*, *int delay*)

declare_sample_delay(xor_bb_sptr self, unsigned int delay)

xor_bb_sptr.**message_subscribers**(*xor_bb_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

xor_bb_sptr.**min_noutput_items**(*xor_bb_sptr self*) → int

xor_bb_sptr.**pc_input_buffers_full_avg**(*xor_bb_sptr self*, *int which*) → float

pc_input_buffers_full_avg(xor_bb_sptr self) -> pmt_vector_float

xor_bb_sptr.**pc_noutput_items_avg**(*xor_bb_sptr self*) → float

xor_bb_sptr.**pc_nproduced_avg**(*xor_bb_sptr self*) → float

xor_bb_sptr.**pc_output_buffers_full_avg**(*xor_bb_sptr self*, *int which*) → float

pc_output_buffers_full_avg(xor_bb_sptr self) -> pmt_vector_float

xor_bb_sptr.**pc_throughput_avg**(*xor_bb_sptr self*) → float

xor_bb_sptr.**pc_work_time_avg**(*xor_bb_sptr self*) → float

xor_bb_sptr.**pc_work_time_total**(*xor_bb_sptr self*) → float

xor_bb_sptr.**sample_delay**(*xor_bb_sptr self*, *int which*) → unsigned int

xor_bb_sptr.**set_min_noutput_items**(*xor_bb_sptr self*, *int m*)

xor_bb_sptr.**set_thread_priority**(*xor_bb_sptr self*, *int priority*) → int

xor_bb_sptr.**thread_priority**(*xor_bb_sptr self*) → int

gnuradio.blocks.**xor_ii**(*size_t vlen=1*) → xor_ii_sptr

output = input_0 ^ input_1 ^ , ... ^ input_N)

Bitwise boolean xor across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

xor_ii_sptr.**active_thread_priority**(*xor_ii_sptr self*) → int

xor_ii_sptr.**declare_sample_delay**(*xor_ii_sptr self*, *int which*, *int delay*)
declare_sample_delay(xor_ii_sptr self, unsigned int delay)

xor_ii_sptr.**message_subscribers**(*xor_ii_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

xor_ii_sptr.**min_noutput_items**(*xor_ii_sptr self*) → int

xor_ii_sptr.**pc_input_buffers_full_avg**(*xor_ii_sptr self*, *int which*) → float
pc_input_buffers_full_avg(xor_ii_sptr self) -> pmt_vector_float

xor_ii_sptr.**pc_noutput_items_avg**(*xor_ii_sptr self*) → float

xor_ii_sptr.**pc_nproduced_avg**(*xor_ii_sptr self*) → float

xor_ii_sptr.**pc_output_buffers_full_avg**(*xor_ii_sptr self*, *int which*) → float
pc_output_buffers_full_avg(xor_ii_sptr self) -> pmt_vector_float

xor_ii_sptr.**pc_throughput_avg**(*xor_ii_sptr self*) → float

xor_ii_sptr.**pc_work_time_avg**(*xor_ii_sptr self*) → float

xor_ii_sptr.**pc_work_time_total**(*xor_ii_sptr self*) → float

xor_ii_sptr.**sample_delay**(*xor_ii_sptr self*, *int which*) → unsigned int

xor_ii_sptr.**set_min_noutput_items**(*xor_ii_sptr self*, *int m*)

xor_ii_sptr.**set_thread_priority**(*xor_ii_sptr self*, *int priority*) → int

xor_ii_sptr.**thread_priority**(*xor_ii_sptr self*) → int

gnuradio.blocks.**xor_ss**(*size_t vlen=1*) → xor_ss_sptr
output = input_0 ^ input_1 ^ , ... ^ input_N)

Bitwise boolean xor across all input streams.

Constructor Specific Documentation:

Parameters: **vlen** –

xor_ss_sptr.**active_thread_priority**(*xor_ss_sptr self*) → int

xor_ss_sptr.**declare_sample_delay**(*xor_ss_sptr self*, *int which*, *int delay*)
declare_sample_delay(xor_ss_sptr self, unsigned int delay)

xor_ss_sptr.**message_subscribers**(*xor_ss_sptr self*, *swig_int_ptr which_port*) → swig_int_ptr

xor_ss_sptr.**min_noutput_items**(*xor_ss_sptr self*) → int

xor_ss_sptr.**pc_input_buffers_full_avg**(*xor_ss_sptr self*, *int which*) → float
pc_input_buffers_full_avg(xor_ss_sptr self) -> pmt_vector_float

xor_ss_sptr.**pc_noutput_items_avg**(*xor_ss_sptr self*) → float

xor_ss_sptr.**pc_nproduced_avg**(*xor_ss_sptr self*) → float

xor_ss_sptr.**pc_output_buffers_full_avg**(*xor_ss_sptr self*, *int which*) → float
pc_output_buffers_full_avg(xor_ss_sptr self) -> pmt_vector_float

xor_ss_sptr.**pc_throughput_avg**(*xor_ss_sptr self*) → float

xor_ss_sptr.**pc_work_time_avg**(*xor_ss_sptr self*) → float

xor_ss_sptr.**pc_work_time_total**(*xor_ss_sptr self*) → float

xor_ss_sptr.**sample_delay**(*xor_ss_sptr self*, *int which*) → unsigned int

xor_ss_sptr.**set_min_noutput_items**(*xor_ss_sptr self*, *int m*)

xor_ss_sptr.**set_thread_priority**(*xor_ss_sptr self*, *int priority*) → int

xor_ss_sptr.**thread_priority**(*xor_ss_sptr self*) → int