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

gnuradio.pmt

pmt.**acons**(*swig_int_ptr x*, *swig_int_ptr y*, *swig_int_ptr a*) → swig_int_ptr
(acons x y a) == (cons (cons x y) a)

pmt.**any_ref**(*swig_int_ptr obj*) → boost::any
Return underlying boost::any.

pmt.**any_set**(*swig_int_ptr obj*, *boost::any const & any*)
Store in .

pmt.**assoc**(*swig_int_ptr obj*, *swig_int_ptr alist*) → swig_int_ptr
Find the first pair in whose car field is and return that pair.

(for “association list”) must be a list of pairs. If no pair in has as its car then #f is returned. Uses pmt::equal to compare with car fields of the pairs in .

pmt.**assq**(*swig_int_ptr obj*, *swig_int_ptr alist*) → swig_int_ptr
Find the first pair in whose car field is and return that pair.

(for “association list”) must be a list of pairs. If no pair in has as its car then #f is returned. Uses pmt::eq to compare with car fields of the pairs in .

pmt.**assv**(*swig_int_ptr obj*, *swig_int_ptr alist*) → swig_int_ptr
Find the first pair in whose car field is and return that pair.

(for “association list”) must be a list of pairs. If no pair in has as its car then #f is returned. Uses pmt::eqv to compare with car fields of the pairs in .

pmt.**blob_data**(*swig_int_ptr blob*) → void const *
Return a pointer to the blob’s data.

pmt.**blob_length**(*swig_int_ptr blob*) → size_t
Return the blob’s length in bytes.

pmt.**c32vector_elements**(*swig_int_ptr v*) → pmt_vector_cfloat

pmt.**c32vector_ref**(*swig_int_ptr v*, *size_t k*) → std::complex< float >

pmt.**c32vector_set**(*swig_int_ptr v*, *size_t k*, *std::complex< float > x*)

pmt.**c64vector_elements**(*swig_int_ptr v*) → pmt_vector_cdouble

pmt.**c64vector_ref**(*swig_int_ptr v*, *size_t k*) → std::complex< double >

pmt.**c64vector_set**(*swig_int_ptr v*, *size_t k*, *std::complex< double > x*)

pmt.**caar**(*swig_int_ptr pair*) → swig_int_ptr

pmt.**cadddr**(*swig_int_ptr pair*) → swig_int_ptr

pmt.**caddr**(*swig_int_ptr pair*) → swig_int_ptr

pmt.**cadr**(*swig_int_ptr pair*) → swig_int_ptr

pmt.**car**(*swig_int_ptr pair*) → swig_int_ptr
If is a pair, return the car of the , otherwise raise wrong_type.

pmt.**cdar**(*swig_int_ptr pair*) → swig_int_ptr

pmt.**cddr**(*swig_int_ptr pair*) → swig_int_ptr

pmt.**cdr**(*swig_int_ptr pair*) → swig_int_ptr
If is a pair, return the cdr of the , otherwise raise wrong_type.

pmt.**cons**(*swig_int_ptr x*, *swig_int_ptr y*) → swig_int_ptr
Return a newly allocated pair whose car is and whose cdr is .

pmt.**deserialize**(*std::streambuf & source*) → swig_int_ptr
Create obj from portable byte-serial representation.

pmt.**deserialize_str**(*std::string str*) → swig_int_ptr
Provide a simple string generating interface to pmt's deserialize function.

pmt.**dict_add**(*swig_int_ptr dict*, *swig_int_ptr key*, *swig_int_ptr value*) → swig_int_ptr
Return a new dictionary with associated with .

pmt.**dict_delete**(*swig_int_ptr dict*, *swig_int_ptr key*) → swig_int_ptr
Return a new dictionary with removed.

pmt.**dict_has_key**(*swig_int_ptr dict*, *swig_int_ptr key*) → bool
Return true if exists in .

pmt.**dict_items**(*swig_int_ptr dict*) → swig_int_ptr
Return list of (key . value) pairs.

pmt.**dict_keys**(*swig_int_ptr dict*) → swig_int_ptr
Return list of keys.

pmt.**dict_ref**(*swig_int_ptr dict*, *swig_int_ptr key*, *swig_int_ptr not_found*) → swig_int_ptr
If exists in , return associated value; otherwise return .

pmt.**dict_update**(*swig_int_ptr dict1*, *swig_int_ptr dict2*) → swig_int_ptr
Return a new dictionary with k=>v pairs from added.

pmt.**dict_values**(*swig_int_ptr dict*) → swig_int_ptr
Return list of values.

pmt.**dump_sizeof**()

pmt.**eq**(*swig_int_ptr x*, *swig_int_ptr y*) → bool
Return true if x and y are the same object; otherwise return false.

pmt.**equal**(*swig_int_ptr x*, *swig_int_ptr y*) → bool
pmt::equal recursively compares the contents of pairs and vectors, applying pmt::eqv on other objects such as numbers and symbols. pmt::equal may fail to terminate if its arguments are circular data structures.

pmt.**eqv**(*swig_int_ptr x*, *swig_int_ptr y*) → bool
Return true if x and y should normally be regarded as the same object, else false.

pmt.**f32vector_elements**(*swig_int_ptr v*) → pmt_vector_float

pmt.**f32vector_ref**(*swig_int_ptr v*, *size_t k*) → float

pmt.**f32vector_set**(*swig_int_ptr v*, *size_t k*, *float x*)

pmt.**f64vector_elements**(*swig_int_ptr v*) → pmt_vector_double

pmt.**f64vector_ref**(*swig_int_ptr v*, *size_t k*) → double

pmt.**f64vector_set**(*swig_int_ptr v*, *size_t k*, *double x*)

pmt.**from_bool**(*bool val*) → swig_int_ptr
Return #f is val is false, else return #t.

pmt.**from_complex**(*std::complex< double > const & z*) → swig_int_ptr
Return a complex number constructed of the given real and imaginary parts.

pmt.**from_double**(*double x*) → swig_int_ptr
Return the pmt value that represents double .

pmt.**from_float**(*double x*) → swig_int_ptr

pmt.**from_long**(*long x*) → swig_int_ptr
Return the pmt value that represents the integer .

pmt.**from_uint64**(*uint64_t x*) → swig_int_ptr
Return the pmt value that represents the uint64 .

pmt.**get_PMT_EOF**() → swig_int_ptr

pmt.**get_PMT_F**() → swig_int_ptr

pmt.**get_PMT_NIL**() → swig_int_ptr

pmt.**get_PMT_T**() → swig_int_ptr

pmt.**init_c32vector**(*size_t k*, *pmt_vector_cfloat data*) → swig_int_ptr

pmt.**init_c64vector**(*size_t k*, *pmt_vector_cdouble data*) → swig_int_ptr

pmt.**init_f32vector**(*size_t k*, *pmt_vector_float data*) → swig_int_ptr

pmt.**init_f64vector**(*size_t k*, *pmt_vector_double data*) → swig_int_ptr

pmt.**init_s16vector**(*size_t k*, *pmt_vector_int16 data*) → swig_int_ptr

pmt.**init_s32vector**(*size_t k*, *pmt_vector_int32 data*) → swig_int_ptr

pmt.**init_s8vector**(*size_t k*, *pmt_vector_int8 data*) → swig_int_ptr

pmt.**init_u16vector**(*size_t k*, *pmt_vector_uint16 data*) → swig_int_ptr

pmt.**init_u32vector**(*size_t k*, *pmt_vector_uint32 data*) → swig_int_ptr

pmt.**init_u8vector**(*size_t k*, *pmt_vector_uint8 data*) → swig_int_ptr

pmt.**intern**(*std::string const & s*) → swig_int_ptr
Alias for pmt_string_to_symbol.

pmt.**is_any**(*swig_int_ptr obj*) → bool
Return true if is an any.

pmt.**is_blob**(*swig_int_ptr x*) → bool
Return true if is a blob, otherwise false.

pmt.**is_bool**(*swig_int_ptr obj*) → bool
Return true if obj is #t or #f, else return false.

pmt.**is_c32vector**(*swig_int_ptr x*) → bool

pmt.**is_c64vector**(*swig_int_ptr x*) → bool

pmt.**is_complex**(*swig_int_ptr obj*) → bool
return true if is a complex number, false otherwise.

pmt.**is_dict**(*swig_int_ptr obj*) → bool
Return true if is a dictionary (warning: also returns true for a pair)

pmt.**is_eof_object**(*swig_int_ptr obj*) → bool

return true if obj is the EOF object, otherwise return false.

pmt.**is_f32vector**(*swig_int_ptr x*) → bool

pmt.**is_f64vector**(*swig_int_ptr x*) → bool

pmt.**is_false**(*swig_int_ptr obj*) → bool
Return true if obj is #f, else return true.

pmt.**is_integer**(*swig_int_ptr x*) → bool
Return true if is an integer number, else false.

pmt.**is_msg_accepter**(*swig_int_ptr obj*) → bool
Return true if is a msg_accepter.

pmt.**is_null**(*swig_int_ptr x*) → bool
Return true if is the empty list, otherwise return false.

pmt.**is_number**(*swig_int_ptr obj*) → bool
Return true if obj is any kind of number, else false.

pmt.**is_pair**(*swig_int_ptr obj*) → bool
Return true if is a pair, else false (warning: also returns true for a dict)

pmt.**is_real**(*swig_int_ptr obj*) → bool

pmt.**is_s16vector**(*swig_int_ptr x*) → bool

pmt.**is_s32vector**(*swig_int_ptr x*) → bool

pmt.**is_s64vector**(*swig_int_ptr x*) → bool

pmt.**is_s8vector**(*swig_int_ptr x*) → bool

pmt.**is_symbol**(*swig_int_ptr obj*) → bool
Return true if obj is a symbol, else false.

pmt.**is_true**(*swig_int_ptr obj*) → bool
Return false if obj is #f, else return true.

pmt.**is_tuple**(*swig_int_ptr x*) → bool
Return true if is a tuple, otherwise false.

pmt.**is_u16vector**(*swig_int_ptr x*) → bool

pmt.**is_u32vector**(*swig_int_ptr x*) → bool

pmt.**is_u64vector**(*swig_int_ptr x*) → bool

pmt.**is_u8vector**(*swig_int_ptr x*) → bool

pmt.**is_uint64**(*swig_int_ptr x*) → bool
Return true if is an uint64 number, else false.

pmt.**is_uniform_vector**(*swig_int_ptr x*) → bool
true if is any kind of uniform numeric vector

pmt.**is_vector**(*swig_int_ptr x*) → bool
Return true if is a vector, otherwise false.

pmt.**length**(*swig_int_ptr v*) → size_t
Return the number of elements in v.

pmt.**list1**(*swig_int_ptr x1*) → swig_int_ptr

Return a list of length 1 containing .

pmt.**list2**(*swig_int_ptr x1*, *swig_int_ptr x2*) → swig_int_ptr
Return a list of length 2 containing , .

pmt.**list3**(*swig_int_ptr x1*, *swig_int_ptr x2*, *swig_int_ptr x3*) → swig_int_ptr
Return a list of length 3 containing , , .

pmt.**list4**(*swig_int_ptr x1*, *swig_int_ptr x2*, *swig_int_ptr x3*, *swig_int_ptr x4*) → swig_int_ptr
Return a list of length 4 containing , , , .

pmt.**list5**(*swig_int_ptr x1*, *swig_int_ptr x2*, *swig_int_ptr x3*, *swig_int_ptr x4*, *swig_int_ptr x5*) → swig_int_ptr
Return a list of length 5 containing , , , , .

pmt.**list6**(*swig_int_ptr x1*, *swig_int_ptr x2*, *swig_int_ptr x3*, *swig_int_ptr x4*, *swig_int_ptr x5*, *swig_int_ptr x6*) → swig_int_ptr
Return a list of length 6 containing , , , , , .

pmt.**list_add**(*swig_int_ptr list*, *swig_int_ptr item*) → swig_int_ptr
Return with added to it.

pmt.**list_has**(*swig_int_ptr list*, *swig_int_ptr item*) → bool
Return bool of contains .

pmt.**list_rm**(*swig_int_ptr list*, *swig_int_ptr item*) → swig_int_ptr
Return with removed from it.

pmt.**make_any**(*boost::any const & any*) → swig_int_ptr
make an any

pmt.**make_blob**(*void const * buf*, *size_t len*) → swig_int_ptr
Make a blob given a pointer and length in bytes.

The data is copied into the blob.

pmt.**make_c32vector**(*size_t k*, *std::complex< float > fill*) → swig_int_ptr

pmt.**make_c64vector**(*size_t k*, *std::complex< double > fill*) → swig_int_ptr

pmt.**make_dict**() → swig_int_ptr
Make an empty dictionary.

pmt.**make_f32vector**(*size_t k*, *float fill*) → swig_int_ptr

pmt.**make_f64vector**(*size_t k*, *double fill*) → swig_int_ptr

pmt.**make_msg_accepter**(*boost::shared_ptr< gr::messages::msg_accepter > ma*) → swig_int_ptr
make a msg_accepter

pmt.**make_rectangular**(*double re*, *double im*) → swig_int_ptr
Return a complex number constructed of the given real and imaginary parts.

pmt.**make_s16vector**(*size_t k*, *int16_t fill*) → swig_int_ptr

pmt.**make_s32vector**(*size_t k*, *int32_t fill*) → swig_int_ptr

pmt.**make_s64vector**(*size_t k*, *int64_t fill*) → swig_int_ptr

pmt.**make_s8vector**(*size_t k*, *int8_t fill*) → swig_int_ptr

pmt.**make_tuple**() → swig_int_ptr

make_tuple(swig_int_ptr e0) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2, swig_int_ptr e3) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2, swig_int_ptr e3, swig_int_ptr e4) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2, swig_int_ptr e3, swig_int_ptr e4, swig_int_ptr e5) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2, swig_int_ptr e3, swig_int_ptr e4, swig_int_ptr e5, swig_int_ptr e6) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2, swig_int_ptr e3, swig_int_ptr e4, swig_int_ptr e5, swig_int_ptr e6, swig_int_ptr e7) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2, swig_int_ptr e3, swig_int_ptr e4, swig_int_ptr e5, swig_int_ptr e6, swig_int_ptr e7, swig_int_ptr e8) -> swig_int_ptr make_tuple(swig_int_ptr e0, swig_int_ptr e1, swig_int_ptr e2, swig_int_ptr e3, swig_int_ptr e4, swig_int_ptr e5, swig_int_ptr e6, swig_int_ptr e7, swig_int_ptr e8, swig_int_ptr e9) -> swig_int_ptr

pmt.**make_u16vector**(*size_t k*, *uint16_t fill*) → swig_int_ptr

pmt.**make_u32vector**(*size_t k*, *uint32_t fill*) → swig_int_ptr

pmt.**make_u64vector**(*size_t k*, *uint64_t fill*) → swig_int_ptr

pmt.**make_u8vector**(*size_t k*, *uint8_t fill*) → swig_int_ptr

pmt.**make_vector**(*size_t k*, *swig_int_ptr fill*) → swig_int_ptr

Make a vector of length , with initial values set to .

pmt.**map**(*pmt::pmt_t (*)(pmt::pmt_t const &) proc*, *swig_int_ptr list*) → swig_int_ptr

Apply element-wise to the elements of list and returns a list of the results, in order.

must be a list. The dynamic order in which is applied to the elements of is unspecified.

pmt.**member**(*swig_int_ptr obj*, *swig_int_ptr list*) → swig_int_ptr

Return the first sublist of whose car is . If does not occur in , then #f is returned. pmt::member use pmt::equal to compare with the elements of .

pmt.**memq**(*swig_int_ptr obj*, *swig_int_ptr list*) → swig_int_ptr

Return the first sublist of whose car is . If does not occur in , then #f is returned. pmt::memq use pmt::eq to compare with the elements of .

pmt.**memv**(*swig_int_ptr obj*, *swig_int_ptr list*) → swig_int_ptr

Return the first sublist of whose car is . If does not occur in , then #f is returned. pmt::memv use pmt::eqv to compare with the elements of .

pmt.**msg_accepter_ref**(*swig_int_ptr obj*) → boost::shared_ptr< gr::messages::msg_accepter >

Return underlying msg_accepter.

pmt.**nth**(*size_t n*, *swig_int_ptr list*) → swig_int_ptr

locates element of list where the car is the ‘zeroth’ element.

pmt.**nthcdr**(*size_t n*, *swig_int_ptr list*) → swig_int_ptr

returns the tail of that would be obtained by calling cdr times in succession.

pmt.**pmt_vector_cdouble**(**args*)

Proxy of C++ std::vector<(std::complex<(double)>)> class.

pmt.**pmt_vector_cfloat**(**args*)

Proxy of C++ std::vector<(std::complex<(float)>)> class.

pmt.**pmt_vector_double**(**args*)

Proxy of C++ std::vector<(double)> class.

pmt.**pmt_vector_float**(**args*)

Proxy of C++ std::vector<(float)> class.

pmt.**pmt_vector_int16**(**args*)
Proxy of C++ std::vector<(int16_t)> class.

pmt.**pmt_vector_int32**(**args*)
Proxy of C++ std::vector<(int32_t)> class.

pmt.**pmt_vector_int8**(**args*)
Proxy of C++ std::vector<(int8_t)> class.

pmt.**pmt_vector_uint16**(**args*)
Proxy of C++ std::vector<(uint16_t)> class.

pmt.**pmt_vector_uint32**(**args*)
Proxy of C++ std::vector<(uint32_t)> class.

pmt.**pmt_vector_uint8**(**args*)
Proxy of C++ std::vector<(uint8_t)> class.

pmt.**read**(*std::istream & port*) → swig_int_ptr
read converts external representations of pmt objects into the objects themselves. Read returns the next object parsable from the given input port, updating port to point to the first character past the end of the external representation of the object.

If an end of file is encountered in the input before any characters are found that can begin an object, then an end of file object is returned. The port remains open, and further attempts to read will also return an end of file object. If an end of file is encountered after the beginning of an object's external representation, but the external representation is incomplete and therefore not parsable, an error is signaled.

pmt.**reverse**(*swig_int_ptr list*) → swig_int_ptr
reverse .

must be a proper list.

pmt.**reverse_x**(*swig_int_ptr list*) → swig_int_ptr
destructively reverse .

must be a proper list.

pmt.**s16vector_elements**(*swig_int_ptr v*) → pmt_vector_int16

pmt.**s16vector_ref**(*swig_int_ptr v*, *size_t k*) → int16_t

pmt.**s16vector_set**(*swig_int_ptr v*, *size_t k*, *int16_t x*)

pmt.**s32vector_elements**(*swig_int_ptr v*) → pmt_vector_int32

pmt.**s32vector_ref**(*swig_int_ptr v*, *size_t k*) → int32_t

pmt.**s32vector_set**(*swig_int_ptr v*, *size_t k*, *int32_t x*)

pmt.**s64vector_ref**(*swig_int_ptr v*, *size_t k*) → int64_t

pmt.**s64vector_set**(*swig_int_ptr v*, *size_t k*, *int64_t x*)

pmt.**s8vector_elements**(*swig_int_ptr v*) → pmt_vector_int8

pmt.**s8vector_ref**(*swig_int_ptr v*, *size_t k*) → int8_t

pmt.**s8vector_set**(*swig_int_ptr v*, *size_t k*, *int8_t x*)

pmt.**serialize**(*swig_int_ptr obj*, *std::streambuf & sink*) → bool
Write portable byte-serial representation of to .

pmt.**serialize_str**(*swig_int_ptr obj*) → std::string

Provide a simple string generating interface to pmt's serialize function.

pmt.**set_car**(*swig_int_ptr pair*, *swig_int_ptr value*)

Stores in the car field of .

pmt.**set_cdr**(*swig_int_ptr pair*, *swig_int_ptr value*)

Stores in the cdr field of .

pmt.**string_to_symbol**(*std::string const & s*) → swig_int_ptr

Return the symbol whose name is .

pmt.**subsetp**(*swig_int_ptr list1*, *swig_int_ptr list2*) → bool

Return true if every element of appears in , and false otherwise. Comparisons are done with pmt::eqv.

pmt.**symbol_to_string**(*swig_int_ptr sym*) → std::string const

If a symbol, return the name of the symbol as a string. Otherwise, raise the wrong_type exception.

pmt.**to_bool**(*swig_int_ptr val*) → bool

Return true if val is pmt::True, return false when val is pmt::PMT_F,.

pmt.**to_complex**(*swig_int_ptr z*) → std::complex< double >

If is complex, real or integer, return the closest complex<double>. Otherwise, raise the wrong_type exception.

pmt.**to_double**(*swig_int_ptr x*) → double

Convert pmt to double if possible.

Returns the number closest to that is representable as a double. The argument must be a real or integer, otherwise a wrong_type exception is raised.

pmt.**to_float**(*swig_int_ptr x*) → double

Convert pmt to float if possible.

This basically is to_double() with a type-cast; the PMT stores the value as a double in any case. Use this when strict typing is required.

pmt.**to_long**(*swig_int_ptr x*) → long

Convert pmt to long if possible.

When represents an exact integer that fits in a long, return that integer. Else raise an exception, either wrong_type when x is not an exact integer, or out_of_range when it doesn't fit.

pmt.**to_pmt**(*p*)

pmt.**to_python**(*p*)

pmt.**to_tuple**(*swig_int_ptr x*) → swig_int_ptr

If is a vector or proper list, return a tuple containing the elements of x

pmt.**to_uint64**(*swig_int_ptr x*) → uint64_t

Convert pmt to uint64 if possible.

When represents an exact integer that fits in a uint64, return that uint64. Else raise an exception, either wrong_type when x is not an exact uint64, or out_of_range when it doesn't fit.

pmt.**tuple_ref**(*swig_int_ptr tuple*, *size_t k*) → swig_int_ptr

Return the contents of position of . must be a valid index of .

pmt.**u16vector_elements**(*swig_int_ptr v*) → pmt_vector_uint16

pmt.**u16vector_ref**(*swig_int_ptr v*, *size_t k*) → uint16_t

pmt.**u16vector_set**(*swig_int_ptr v*, *size_t k*, *uint16_t x*)

pmt.**u32vector_elements**(*swig_int_ptr v*) → pmt_vector_uint32

pmt.**u32vector_ref**(*swig_int_ptr v*, *size_t k*) → uint32_t

pmt.**u32vector_set**(*swig_int_ptr v*, *size_t k*, *uint32_t x*)

pmt.**u64vector_ref**(*swig_int_ptr v*, *size_t k*) → uint64_t

pmt.**u64vector_set**(*swig_int_ptr v*, *size_t k*, *uint64_t x*)

pmt.**u8vector_elements**(*swig_int_ptr v*) → pmt_vector_uint8

pmt.**u8vector_ref**(*swig_int_ptr v*, *size_t k*) → uint8_t

pmt.**u8vector_set**(*swig_int_ptr v*, *size_t k*, *uint8_t x*)

pmt.**uniform_vector_elements**(*swig_int_ptr v*, *size_t & len*) → void const *

pmt.**uniform_vector_itemsize**(*swig_int_ptr x*) → size_t
item size in bytes if is any kind of uniform numeric vector

pmt.**vector_fill**(*swig_int_ptr vector*, *swig_int_ptr fill*)
Store in every position of .

pmt.**vector_ref**(*swig_int_ptr vector*, *size_t k*) → swig_int_ptr
Return the contents of position of . must be a valid index of .

pmt.**vector_set**(*swig_int_ptr vector*, *size_t k*, *swig_int_ptr obj*)
Store in position .

pmt.**write**(*swig_int_ptr obj*, *std::ostream & port*)
Write a written representation of to the given .

pmt.**write_string**(*swig_int_ptr obj*) → std::string
Return a string representation of . This is the same output as would be generated by pmt::write.