pMath

Generated by Doxygen 1.5.9

Thu Sep 6 00:43:12 2012

Contents

1 The pMath Computer Algebra System Library

Author:

Peter Frentrup

Date:

2011

Introduction

pMath is a free CAS for Windows and Unix like systems. The whole CAS consists of three projects:

- The pMath library documented here, which implements the parser, interpreter, mathematical functionality and OS binding.
- The RichMath graphical front-end.
- Addon libraries/modules for the pMath library and Language (e.g. a Java binding).

You as a user (front-end or module programmer) of the pMath library just have to #include <pmath.h> and link with the appropriate library file.

This document does not cover the pMath *Language* itself.

Links/Depencies

pMath is build on top of several open source libraries:

- GMP (`http://gmplib.org`)
- MPFR (`http://www.mpfr.org`)
- PCRE (`http://www.pcre.org`)

2 Todo List

Class pmath_thread_t Implement pmath_run_parallel(number_of_parallel_threads, callback).

Global pmath_task_t document pmath-util/concurrency/threadpool.h

Global pmath_thread_send_wait Check, what happens if mq belongs to a parent thread.

3 Deprecated List

Global pmath_symbol_t::pmath_symbol_get_value(pmath_symbol_t symbol)

Global pmath_symbol_t::pmath_symbol_set_value(pmath_symbol_t symbol, pmath_t value)

Global pmath_symbol_t::pmath_symbol_synchronized(pmath_symbol_t symbol, pmath_callback_t callback, v

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Namespace Index

5.1 Namespace List

Here is a list of all namespaces with brief descriptions:

6 Data Structure Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

7 Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

8 File Index

8.1 File List

Here is a list of all files with brief descriptions:

9 Module Documentation

9.1 Custom Objects

Encapsulate arbitrary data in pMath objects.

Data Structures

- class pmath_custom_t

  The Custom Object class.

Typedefs

- typedef pmath_t pmath_custom_t

Functions

- pmath_custom_t pmath_custom_new (void ∗data, pmath_callback_t destructor)

  Create a custom object.

- void ∗ pmath_custom_get_data (pmath_custom_t custom)

  Get a custom object's data member.

- pmath_bool_t pmath_custom_has_destructor (pmath_custom_t custom, pmath_callback_t dtor)

  Check for a custom object's data type.

9.1.1 Detailed Description

Encapsulate arbitrary data in pMath objects.

Custom Objects consist of a pointer and a destructor. The destructor is called (with the pointer as its argument) when the custom object's reference pointer yields zero.

Custom Objects are not evaluateable. This means evaluation of such an object returns PMATH_NULL. But you can store custom objects in symbols (directly with pmath_symbol_set_value()).

A symbol that holds a custom object remains unevaluated. It can also contain function definitions. But those must be set *after* setting the value with pmath_symbol_set_value(my_symbol, my_custom_object):

Example: You want to store a custom object and a function definition in a symbol (my_symbol/: answer(my_symbol):= 42).

```
pmath_custom_t my_custom_object = pmath_custom_new(my_data, my_destructor);
pmath_symbol_set_value(my_symbol, my_custom_object);

pmath_unref(pmath_evaluate(
  pmath_parse_string("`1`/: answer(`1`):= 42", 1, pmath_ref(my_symbol))));
```

9.1.2 Typedef Documentation

9.1.2.1 typedef pmath_t pmath_custom_t

9.1.3 Function Documentation

9.1.3.1 void ∗ pmath_custom_get_data (pmath_custom_t *custom*) `[inherited]`

Get a custom object's data member.

Parameters:

custom A custom object.

Returns:

The objects data member or PMATH_NULL if *custom* is PMATH_NULL.

Note that you cannot assume anything about the content of this pointer unless you know its destructor (check pmath_custom_has_destructor).

All access to ∗data must be threadsafe/synchronized. By convention, you are the only one who moves custom objects with your destructor around (other modules should not handle custom objects whose destructor they do not know). And normally, each of your custom objects is stored in one symbol. So synchronization can be done with pmath_symbol_synchronized(). If one of these conditions is not met and a custom object could be accessd from multiple threads (See Multithreading with pMath), you must also store a synchronization object (e.g. symbol or threadlock) in the *data* member und use this.

9.1.3.2 pmath_bool_t pmath_custom_has_destructor (pmath_custom_t *custom*, pmath_callback_t *dtor*) `[inherited]`

Check for a custom object's data type.

Parameters:

custom A custom object.

dtor A callback function.

Returns:

TRUE if the object's destructor is *dtor*.

9.1.3.3 pmath_custom_t pmath_custom_new (void ∗ *data*, pmath_callback_t *destructor*) `[inherited]`

Create a custom object.

Parameters:

data An arbitrary pointer.

destructor A function that will be called on object destruction to enable freeing of *data*.

Returns:

A custom object or PMATH_NULL on failure (in that case, *destructor(data)* is called immediately).

9.2 Expressions

Expression objects in pMath.

Data Structures

- class pmath_expr_t

  The Expression class.

Typedefs

- typedef pmath_t pmath_expr_t

Functions

- pmath_expr_t pmath_expr_new (pmath_t head, size_t length)

  Create a new expression.

- pmath_expr_t pmath_expr_new_extended (pmath_t head, size_t length,...)

  Create a new expression with all items given.

- pmath_expr_t pmath_expr_resize (pmath_expr_t expr, size_t new_length)

  Resize an expression.

- pmath_expr_t pmath_expr_append (pmath_expr_t expr, size_t count,...)

  Append some items to an expression.

- size_t pmath_expr_length (pmath_expr_t expr)

  Get an expression's length.

- pmath_t pmath_expr_get_item (pmath_expr_t expr, size_t index)

  Get an item from an expression.

- pmath_t pmath_expr_extract_item (pmath_expr_t expr, size_t index)

  Extract an item from an expression.

- pmath_expr_t pmath_expr_get_item_range (pmath_expr_t expr, size_t start, size_t length)

  Get multiple items from an expression.

- const pmath_t * pmath_expr_read_item_data (pmath_expr_t expr)

  Get a pointer to the expression's internal items array.

- pmath_expr_t pmath_expr_set_item (pmath_expr_t expr, size_t index, pmath_t item)

  Set an item in an expression.

- pmath_expr_t pmath_expr_remove_all (pmath_expr_t expr, pmath_t rem)

  Remove all occurencies of an object from an expression.

- pmath_expr_t pmath_expr_sort (pmath_expr_t expr)

  Sort an expression.

- pmath_expr_t pmath_expr_flatten (pmath_expr_t expr, pmath_t head, size_t depth)

  Flatten an expression.

9.2.1 Detailed Description

Expression objects in pMath.

Any pMath language-level expression (lists, terms, function calls, ...) is stored in a pmath_expr_t – an array of pMath objects.

See also:

Object Utility Functions

9.2.2 Typedef Documentation

9.2.2.1 typedef pmath_t pmath_expr_t

9.2.3 Function Documentation

9.2.3.1 pmath_expr_t pmath_expr_append (pmath_expr_t *expr*, size_t *count*, ...) `[inherited]`

Append some items to an expression.

Parameters:

expr The old expression. It will be freed/invalid after the call.

count The number of items to append.

... exactly count pmath_ts. Do not use them after the call.

Returns:

PMATH_NULL or a new expression that contains all items of expr followed by the items in '...'. You must destroy it with pmath_unref().

If expr == PMATH_NULL, the returns value is pmath_expr_new_extended(PMATH_NULL,count,...).

9.2.3.2 pmath_t pmath_expr_extract_item (pmath_expr_t *expr*, size_t *index*) `[inherited]`

Extract an item from an expression.

Parameters:

expr A pMath expression.

index The index of the item.

Returns:

Same as pmath_expr_get_item() but if expr has refcount==1, the item might be removed from expr to ensure that the item's refcount is 1.

Normaly, you should use pmath_expr_get_item(). This function if for use in loops which modify items of an expression.

9.2.3.3 pmath_expr_t pmath_expr_flatten (pmath_expr_t *expr*, pmath_t *head*, size_t *depth*) [inherited]

Flatten an expression.

Parameters:

expr A pMath expression. It will be destroyed, do not use it after the call.

head The head of items, that should be flattened out. It will be destroyed, so you can use 'pmath_expr_get_item(expr)' directly to use expr's head.

depth The depth to which level flattening should be done. A value of 0 means 'no flattening'.

Returns:

A new expression where all items, that have the same head as expr, will be flattened.

9.2.3.4 pmath_t pmath_expr_get_item (pmath_expr_t *expr*, size_t *index*) [inherited]

Get an item from an expression.

Parameters:

expr A pMath expression.

index The index of the item.

Returns:

A copy of the requested item, if index is not greater than the length of expr and PMATH_NULL otherwise. You must destroy it with pmath_unref().

9.2.3.5 pmath_expr_t pmath_expr_get_item_range (pmath_expr_t *expr*, size_t *start*, size_t *length*) `[inherited]`

Get multiple items from an expression.

Parameters:

- ***expr*** A pMath expression. It will ∗not∗ be destroyed.
- ***start*** The start index of the items.
- ***length*** The number of the items.

Returns:

A new expression with the same head as *expr*. Its length is max(0, min(start + length, 1 + pmath_expr_length(expr)) - start) and it contains the items from expr beginning at index *start*.

9.2.3.6 size_t pmath_expr_length (pmath_expr_t *expr*) `[inherited]`

Get an expression's length.

Parameters:

- ***expr*** A pMath expression.

Returns:

The number of items in expr (not counting the head).

9.2.3.7 pmath_expr_t pmath_expr_new (pmath_t *head*, size_t *length*) `[inherited]`

Create a new expression.

Parameters:

- ***head*** The expression's head (index 0). Do not use them after the call.
- ***length*** The number of additional items in the expression.

Returns:

PMATH_NULL or a new expression with head at index 0 and all other items initialized to PMATH_NULL. You must destroy it with pmath_unref().

9.2.3.8 pmath_expr_t pmath_expr_new_extended (pmath_t *head*, size_t *length*, ...) [inherited]

Create a new expression with all items given.

Parameters:

head The expression's head (index 0). Do not use them after the call.

length The number of additional items in the expression.

... exactly length pmath_ts. Do not use them after the call.

Returns:

PMATH_NULL or a new expression with head at index 0 all items at index i = 1..length initialized to the i'th argument in '...'. You must destroy it with pmath_unref().

9.2.3.9 const pmath_t ∗ pmath_expr_read_item_data (pmath_expr_t *expr*) [inherited]

Get a pointer to the expression's internal items array.

Parameters:

expr A pMath expression. It will ∗not∗ be destroyed.

Returns:

A 0-based array of pmath_t or NULL on error. This array is only valid while *expr* is valid and not changed.

This function is for fast reading access to multiple items. You have to do all the error checking alone. Note that result[0] === pmath_expr_get_item(expr, 1), a.s.o.

9.2.3.10 pmath_expr_t pmath_expr_remove_all (pmath_expr_t *expr*, pmath_t *rem*) [inherited]

Remove all occurencies of an object from an expression.

Parameters:

expr A pMath expression. It will be destroyed, do not use it after the call.

rem The object to be removed. It will ∗not∗ be destroyed.

Returns:

PMATH_NULL or a new expression that contains no occurencies of *rem* (except maybe the head). It is a shrinked version of *expr*.

9.2.3.11 pmath_expr_t pmath_expr_resize (pmath_expr_t *expr*, size_t *new_length*) [inherited]

Resize an expression.

Parameters:

expr The old expression. It will be freed/invalid after the call.

new_length The new length of the expression.

Returns:

PMATH_NULL or a new expression of length new_length. You must destroy it with pmath_unref().

If expr's length is less than or equals to new_length, all items at 1..(expr's length) are be copied and the rest is initialized with PMATH_NULL. Otherwise, all items at 1..new_length are copied, those at (new_length+1)..(expr's length) are be freed and the rest is initialized with PMATH_NULL.

9.2.3.12 pmath_expr_t pmath_expr_set_item (pmath_expr_t *expr*, size_t *index*, pmath_t *item*) [inherited]

Set an item in an expression.

Parameters:

expr A pMath expression. It will be destroyed, do not use it after the call.

index The index of the to-be-changed item.

item The new value of the item. It will be destroyed, do not use it after the call.

Returns:

PMATH_NULL or a new expression with item at index. You must destroy it with pmath_unref().

If index is greater than expr's length, item will be destroyed and the return value is expr.

9.2.3.13 pmath_expr_t pmath_expr_sort (pmath_expr_t *expr*) [inherited]

Sort an expression.

Parameters:

expr A pMath expression. It will be destroyed, do not use it after the call.

Returns:

A new expression where all items from expr are sorted (except the head, which remains unchanged).

9.3 Numbers

Number objects in pMath.

Data Structures

- class pmath_number_t

  The abstract Number class.

- class pmath_rational_t

  The abstract Rational Number class.

- class pmath_integer_t

  The Integer class.

- class pmath_quotient_t

  The Quotient class.

- class pmath_float_t

  The Floating Point Number class.

Defines

- #define PMATH_MACHINE_PRECISION 0
- #define PMATH_AUTO_PRECISION 1
- #define pmath_integer_new_si32(si) PMATH_FROM_INT32(si)

  Create an integer object from an int32_t.

- #define pmath_integer_new_siptr(si)

  Create an integer object from an intptr_t.

- #define pmath_integer_new_uiptr(ui)

  Create an integer object from an uintptr_t.

- #define pmath_integer_fits_si32(integer) pmath_is_int32(integer)

  Check whether a pMath integer is in range -2^31 .. 2^31-1.

- #define pmath_integer_fits_siptr(integer)

  Check whether a pMath integer fits into an intptr_t.

- #define pmath_integer_fits_uiptr(integer)

  Check whether a pMath integer fits into an uintptr_t.

- #define pmath_integer_get_siptr

  Convert a pMath integer to a intptr_t.

- #define pmath_integer_get_uiptr

  Convert a pMath integer to a uintptr_t.

- pmath_integer_new_si32(si)

  Create an integer object from an int32_t.

- pmath_integer_new_siptr(si)

  Create an integer object from an intptr_t.

- pmath_integer_new_uiptr(ui)

  Create an integer object from an uintptr_t.

- pmath_integer_fits_si32(integer)

  Check whether a pMath integer is in range -2^31 .. 2^31-1.

- pmath_integer_fits_siptr(integer)

  Check whether a pMath integer fits into an intptr_t.

- pmath_integer_fits_uiptr(integer)

  Check whether a pMath integer fits into an uintptr_t.

- pmath_integer_get_siptr

  Convert a pMath integer to a intptr_t.

- pmath_integer_get_uiptr

  Convert a pMath integer to a uintptr_t.

Typedefs

- typedef pmath_t pmath_number_t
- typedef pmath_number_t pmath_rational_t
- typedef pmath_rational_t pmath_integer_t
- typedef pmath_rational_t pmath_mpint_t
- typedef pmath_rational_t pmath_quotient_t
- typedef pmath_number_t pmath_float_t
- typedef pmath_float_t pmath_mpfloat_t

Enumerations

- enum pmath_precision_control_t {
  PMATH_PREC_CTRL_AUTO = 0, PMATH_PREC_CTRL_MACHINE_PREC = 1,
  PMATH_PREC_CTRL_GIVEN_PREC = 2, PMATH_PREC_CTRL_GIVEN_ACC = 3 }

Functions

- pmath_bool_t pmath_is_numeric (pmath_t obj)

  Test whether an expression is a numeric quantity.

- double pmath_accuracy (pmath_t obj)

  Get the accuracy (in bits) of an object.

- double pmath_precision (pmath_t obj)

  Get the precision (in bits) of an object.

- pmath_t pmath_set_accuracy (pmath_t obj, double acc)

  Set an object's accuracy in bits.

- pmath_t pmath_set_precision (pmath_t obj, double prec)

  Set an object's accuracy in bits.

- pmath_t pmath_approximate (pmath_t obj, double precision_goal, double accuracy_goal, pmath_bool_t ∗aborted)

  Approximate an object.

- pmath_integer_t pmath_integer_new_slong (signed long int si)

  Create an integer object from a signed long.

- pmath_integer_t pmath_integer_new_ulong (unsigned long int ui)

  Create an integer object from an unsigned long.

- pmath_integer_t pmath_integer_new_ui32 (uint32_t ui)

Create an integer object from an uint32_t.

- pmath_integer_t pmath_integer_new_si64 (int64_t si)

  Create an integer object from an int64_t.

- pmath_integer_t pmath_integer_new_ui64 (uint64_t ui)

  Create an integer object from an uint64_t.

- pmath_integer_t pmath_integer_new_data (size_t count, int order, int size, int endian, size_t nails, const void ∗data)

  Create an integer object from a data buffer.

- pmath_integer_t pmath_integer_new_str (const char ∗str, int base)

  Create an integer object from a C String.

- pmath_rational_t pmath_rational_new (pmath_integer_t numerator, pmath_integer_t denominator)

  Create a rational number.

- pmath_integer_t pmath_rational_numerator (pmath_rational_t rational)

  Get the numerator of a rational number.

- pmath_integer_t pmath_rational_denominator (pmath_rational_t rational)

  Get the denominator of a rational number.

- pmath_number_t pmath_float_new_str (const char ∗str, int base, pmath_precision_control_t precision_control, double base_precision_accuracy)

  Create a floating point number from a string.

- pmath_bool_t pmath_integer_fits_ui32 (pmath_integer_t integer)

  Check whether a pMath integer is in range 0 .. 2^32-1.

- pmath_bool_t pmath_integer_fits_si64 (pmath_integer_t integer)

  Check whether a pMath integer is in range -2^63 .. 2^63-1.

- pmath_bool_t pmath_integer_fits_ui64 (pmath_integer_t integer)

  Check whether a pMath integer is in range 0 .. 2^64-1.

- int32_t pmath_integer_get_si32 (pmath_integer_t integer)

  Convert a pMath integer to a signed long int.

- uint32_t pmath_integer_get_ui32 (pmath_integer_t integer)

  Convert a pMath integer to a unsigned long int.

- int64_t pmath_integer_get_si64 (pmath_integer_t integer)

  Convert a pMath integer to an int64_t.

- uint64_t pmath_integer_get_ui64 (pmath_integer_t integer)

  Convert a pMath integer to a uint64_t.

- double pmath_number_get_d (pmath_number_t number)

  Convert a pMath number to a double.

- int pmath_number_sign (pmath_number_t num)

  Get a number's sign.

- pmath_number_t pmath_number_neg (pmath_number_t num)

  Get a number's negative.

9.3.1 Detailed Description

Number objects in pMath.

pMath supports arbitrary big integers and rational values, floating point numbers in machine precision or with automatic precision tracking and complex numbers (the latter are represented by ordinary pmath_expr_t, all other number types have their own internal representation).

Note that in might be more convinient to use pmath_build_value() than the specialized constructors represented here, because the former supports Infinity and Undefined (NaN) values for C `double`s.

The GNU Multiple Precision Library (`http://gmplib.org/`) is used for integer and rational arithmetic and the MPFR library (`http://www.mpfr.org/`) for floating point arithmetic.

9.3.2 Define Documentation

9.3.2.1 #define PMATH_AUTO_PRECISION 1

9.3.2.2 pmath_integer_fits_si32(integer) `[inherited]`

Check whether a pMath integer is in range -2^31 .. 2^31-1.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough for an int32_t.

9.3.2.3 #define pmath_integer_fits_si32(integer) pmath_is_int32(integer)

Check whether a pMath integer is in range -2^31 .. 2^31-1.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough for an int32_t.

9.3.2.4 pmath_integer_fits_siptr(integer) `[inherited]`

Check whether a pMath integer fits into an intptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough.

9.3.2.5 #define pmath_integer_fits_siptr(integer)

Check whether a pMath integer fits into an intptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough.

9.3.2.6 pmath_integer_fits_uiptr(integer) `[inherited]`

Check whether a pMath integer fits into an uintptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough.

9.3.2.7 #define pmath_integer_fits_uiptr(integer)

Check whether a pMath integer fits into an uintptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough.

9.3.2.8 pmath_integer_get_siptr `[inherited]`

Convert a pMath integer to a intptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_siptr

9.3.2.9 #define pmath_integer_get_siptr

Convert a pMath integer to a intptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_siptr

9.3.2.10 pmath_integer_get_uiptr `[inherited]`

Convert a pMath integer to a uintptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_uiptr

9.3.2.11 #define pmath_integer_get_uiptr

Convert a pMath integer to a uintptr_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_uiptr

9.3.2.12 pmath_integer_new_si32(si) `[inherited]`

Create an integer object from an int32_t.

Parameters:

si An int32_t.

Returns:

A pMath integer with the specified value.

9.3.2.13 #define pmath_integer_new_si32(si) PMATH_FROM_INT32(si)

Create an integer object from an int32_t.

Parameters:

si An int32_t.

Returns:

A pMath integer with the specified value.

9.3.2.14 pmath_integer_new_siptr(si) `[inherited]`

Create an integer object from an intptr_t.

Parameters:

si An intptr_t value.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.2.15 #define pmath_integer_new_siptr(si)

Create an integer object from an intptr_t.

Parameters:

si An intptr_t value.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.2.16 pmath_integer_new_uiptr(ui) `[inherited]`

Create an integer object from an uintptr_t.

Parameters:

ui A uintptr_t value.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.2.17 #define pmath_integer_new_uiptr(ui)

Create an integer object from an uintptr_t.

Parameters:

ui A uintptr_t value.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.2.18 #define PMATH_MACHINE_PRECISION 0

9.3.3 Typedef Documentation

9.3.3.1 typedef pmath_number_t pmath_float_t

9.3.3.2 typedef pmath_rational_t pmath_integer_t

9.3.3.3 typedef pmath_float_t pmath_mpfloat_t

9.3.3.4 typedef pmath_rational_t pmath_mpint_t

9.3.3.5 typedef pmath_t pmath_number_t

9.3.3.6 typedef pmath_rational_t pmath_quotient_t

9.3.3.7 typedef pmath_number_t pmath_rational_t

9.3.4 Enumeration Type Documentation

9.3.4.1 enum pmath_precision_control_t

Enumerator:

PMATH_PREC_CTRL_AUTO
PMATH_PREC_CTRL_MACHINE_PREC
PMATH_PREC_CTRL_GIVEN_PREC
PMATH_PREC_CTRL_GIVEN_ACC

9.3.5 Function Documentation

9.3.5.1 double pmath_accuracy (pmath_t *obj*)

Get the accuracy (in bits) of an object.

Parameters:

obj An object. It will be freed.

Returns:

The number of known bits after the decimal point.

HUGE_VAL is given for exact quantities. If *obj* is an expression, the minimum of its items' accuracies is returned.

Note that the builtin function Accuracy() uses base 10, but this function operates on base 2.

9.3.5.2 pmath_t pmath_approximate (pmath_t *obj*, double *precision_goal*, double *accuracy_goal*, pmath_bool_t ∗ *aborted*)

Approximate an object.

Parameters:

obj An object. It will be freed.

precision_goal The requested precision in bits.

accuracy_goal The requested accurarcy in bits.

aborted [out] Whether the approximation was aborted and an N::meprec should be generated. When this is NULL, N::meprec will be generated automatically if necessary.

Returns:

The approximated object.

Use `prec == -HUGE_VAL` or `acc == -HUGE_VAL` for machine precision. Use `acc == HUGE_VAL` if the accuracy is not imporant and use `prec == HUGE_VAL` if the precision is not important.

9.3.5.3 pmath_number_t pmath_float_new_str (const char ∗ *str*, int *base*, pmath_precision_control_t *precision_control*, double *base_precision_accuracy*) `[inherited]`

Create a floating point number from a string.

Parameters:

str A C-string representing the value in a given *base*. It should have the form "ddd.ddd" or simply "ddd". An exponent can be appended with "ennn" or if *base* != 10 "@nnn".

base The base between 2 and 36.

precision_control flag for controling the precision.

base_precision_accuracy given precision or accuracy. depending on the value of the above flag.

Returns:

a new pMath floating point number or PMATH_NULL on error or the integer 0 (see below when this happens).

Remarks:

precision_control may have one of the following values:

- `PMATH_PREC_CTRL_AUTO:`
  The precision is specified by the number of digits given in str. It may result in a pMath machine float, mulit-precision float or integer.
  The value of *base_precision_accuracy* will be ignored.
- `PMATH_PREC_CTRL_MACHINE_PREC:`
  The result is a pMath machine float.
  The value of *base_precision_accuracy* will be ignored.

- PMATH_PREC_CTRL_GIVEN_PREC:

  If the number's value is 0, the *integer* 0 will be returned.

  The precision is given by *base_precision_accuracy* (interpreted in the given base).
- PMATH_PREC_CTRL_GIVEN_ACC:

  base_precision_accuracy specifies the accuracy (the number of known *base* -digits after the point). The precision is calculated appropriately.

For a multiprecision float `x != 0` with absolute error `dx`, `accuracy` and `precision` are:

```
accuracy  = -Log(base, dx)
precision = -Log(base, dx / Abs(x))
```

So `precision = accuracy + Log(base, Abs(x)).`

9.3.5.4 pmath_bool_t pmath_integer_fits_si64 (pmath_integer_t *integer*) [inherited]

Check whether a pMath integer is in range -2^63 .. 2^63-1.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough for an int64_t.

9.3.5.5 pmath_bool_t pmath_integer_fits_ui32 (pmath_integer_t *integer*) [inherited]

Check whether a pMath integer is in range 0 .. 2^32-1.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough for an uint32_t.

9.3.5.6 pmath_bool_t pmath_integer_fits_ui64 (pmath_integer_t *integer*) `[inherited]`

Check whether a pMath integer is in range 0 .. 2^64-1.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough for an uint64_t.

9.3.5.7 int32_t pmath_integer_get_si32 (pmath_integer_t *integer*) `[inherited]`

Convert a pMath integer to a signed long int.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_si32

9.3.5.8 int64_t pmath_integer_get_si64 (pmath_integer_t *integer*) `[inherited]`

Convert a pMath integer to an int64_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_si32

9.3.5.9 uint32_t pmath_integer_get_ui32 (pmath_integer_t *integer*) [inherited]

Convert a pMath integer to a unsigned long int.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_ui32

9.3.5.10 uint64_t pmath_integer_get_ui64 (pmath_integer_t *integer*) [inherited]

Convert a pMath integer to a uint64_t.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

The integer's value if it fits.

See also:

pmath_integer_fits_ui32

9.3.5.11 pmath_integer_t pmath_integer_new_data (size_t *count*, int *order*, int *size*, int *endian*, size_t *nails*, const void ∗ *data*) [inherited]

Create an integer object from a data buffer.

Parameters:

count The number of words to be read.

order The order of the words: 1 for most significant word first or -1 for least significant first.

size The size (in bytes) of a word.

endian The byte order within each word: 1 for most significant byte first, -1 for least significant first, or 0 for the native endianness of the CPU.

nails The most significant *nails* bits of each word are skipped. This can be 0 to use the full words.

data The buffer to read from.

Returns:

A non-negative integer.

See also:

GMP's mpz_import()

9.3.5.12 pmath_integer_t pmath_integer_new_si64 (int64_t *si*) `[inherited]`

Create an integer object from an int64_t.

Parameters:

si An int64_t value.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.5.13 pmath_integer_t pmath_integer_new_slong (signed long int *si*) `[inherited]`

Create an integer object from a signed long.

Parameters:

si A signed long int.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.5.14 pmath_integer_t pmath_integer_new_str (const char * *str*, int *base*) `[inherited]`

Create an integer object from a C String.

Parameters:

str A string representing the value in base *base*.

base The base.

Returns:

A pMath integer with the specified value or PMATH_NULL.

See GMP's mpz_set_str for mor information about the parameters.

9.3.5.15 pmath_integer_t pmath_integer_new_ui32 (uint32_t *ui*) `[inherited]`

Create an integer object from an uint32_t.

Parameters:

ui An uint32_t

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.5.16 pmath_integer_t pmath_integer_new_ui64 (uint64_t *ui*) `[inherited]`

Create an integer object from an uint64_t.

Parameters:

ui A uint64_t value.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.5.17 pmath_integer_t pmath_integer_new_ulong (unsigned long int *ui*) `[inherited]`

Create an integer object from an unsigned long.

Parameters:

ui An unsigned long int.

Returns:

A pMath integer with the specified value or PMATH_NULL.

9.3.5.18 pmath_bool_t pmath_is_numeric (pmath_t *obj*)

Test whether an expression is a numeric quantity.

Parameters:

obj An object. It wont be freed.

Returns:

Whether calling pmath_approximate() may return an appxoximate floating point number.

9.3.5.19 double pmath_number_get_d (pmath_number_t *number*) `[inherited]`

Convert a pMath number to a double.

Parameters:

number A pMath number. It wont be freed.

Returns:

The number's value if it fits.

9.3.5.20 pmath_number_t pmath_number_neg (pmath_number_t *num*) `[inherited]`

Get a number's negative.

Parameters:

num A pMath number. It will be freed, do not use it afterwards.

Returns:

-num

9.3.5.21 int pmath_number_sign (pmath_number_t *num*) `[inherited]`

Get a number's sign.

Parameters:

> ***num*** A pMath number. It wont be freed.

Returns:

> The number's sign (-1, 0 or 1)

9.3.5.22 double pmath_precision (pmath_t *obj*)

Get the precision (in bits) of an object.

Parameters:

> ***obj*** An object. It will be freed.

Returns:

> The number of known bits.

HUGE_VAL is given for exact quantities. -HUGE_VAL means "machine precision". If *obj* is an expression, the minimum of its items' accuracies is returned.

Note that the builtin function Precision() uses base 10, but this function operates on base 2.

9.3.5.23 pmath_integer_t pmath_rational_denominator (pmath_rational_t *rational*) `[inherited]`

Get the denominator of a rational number.

Parameters:

> ***rational*** A rational number (integer or quotient). It wont be freed.

Returns:

> A reference to the denominator of *rational* if it is a quotient or 1 if it is an integer. You have to destroy the result e.g. with pmath_unref().

9.3.5.24 pmath_rational_t pmath_rational_new (pmath_integer_t *numerator*, pmath_integer_t *denominator*) `[inherited]`

Create a rational number.

Parameters:

> ***numerator*** The quotient's numerator. It will be freed.
>
> ***denominator*** The quotient's denominator. It will be freed.

Returns:

> An integer, if *denominator* divides *numerator* or a quotient in canonical form otherwise. If denominator is zero, PMATH_NULL will be returned.

9.3.5.25 pmath_integer_t pmath_rational_numerator (pmath_rational_t *rational*) `[inherited]`

Get the numerator of a rational number.

Parameters:

> ***rational*** A rational number (integer or quotient). It wont be freed.

Returns:

> A reference to the numerator of *rational* if it is a quotient or *rational* itself if it is an integer. You have to destroy the result e.g. with pmath_unref().

9.3.5.26 pmath_t pmath_set_accuracy (pmath_t *obj*, double *acc*)

Set an object's accuracy in bits.

Parameters:

> ***obj*** An object. It will be freed.
>
> ***acc*** The new number of known bits after the decimal point.

Returns:

> The new object.

Use `acc == -HUGE_VAL` for machine precision and `prec == -HUGE_VAL` if you want to convert all floating point numbers to exact rational numbers.

Note that the builtin function SetAccuracy() uses base 10, but this function operates on base 2.

9.3.5.27 pmath_t pmath_set_precision (pmath_t *obj*, double *prec*)

Set an object's accuracy in bits.

Parameters:

obj An object. It will be freed.

prec The new number of known bits.

Returns:

The new object.

Use `prec == -HUGE_VAL` for machine precision and `prec == -HUGE_VAL` if you want to convert all floating point numbers to exact rational numbers.

Note that the builtin function SetPrecision() uses base 10, but this function operates on base 2.

9.4 Objects - the Base of pMath

The basic class for all pMath objects.

Data Structures

- class pmath_t

  The basic type of all pMath objects.

- struct pmath_write_ex_t

  Command structure for pmath_write_ex(). This should be inistialized with `memset(&ex, 0, sizeof(ex)); ex.size = sizeof(ex); ...` *.*

Defines

- #define PMATH_TAGMASK_BITCOUNT 12
- #define PMATH_TAGMASK_NONDOUBLE 0x7FF00000U
- #define PMATH_TAGMASK_POINTER 0xFFF00000U
- #define PMATH_TAG_INVALID (PMATH_TAGMASK_NONDOUBLE | 0xFFFFF)
- #define PMATH_TAG_MAGIC (PMATH_TAGMASK_NONDOUBLE | 0x10000)
- #define PMATH_TAG_INT32 (PMATH_TAGMASK_NONDOUBLE | 0x20000)
- #define PMATH_TAG_STR0 (PMATH_TAGMASK_NONDOUBLE | 0x30000)

- #define PMATH_TAG_STR1 (PMATH_TAGMASK_NONDOUBLE | 0x40000)
- #define PMATH_TAG_STR2 (PMATH_TAGMASK_NONDOUBLE | 0x50000)
- #define PMATH_THREAD_KEY_PARSESYMBOLS PMATH_FROM_TAG(PMATH_TAG_MAGIC, 252)
- #define PMATH_THREAD_KEY_PARSERARGUMENTS PMATH_FROM_TAG(PMATH_TAG_MAGIC, 253)
- #define PMATH_ABORT_EXCEPTION PMATH_FROM_TAG(PMATH_TAG_MAGIC, 254)
- #define PMATH_STATIC_UNDEFINED { (((uint64_t)PMATH_TAG_MAGIC) << 32) | 255 }
- #define PMATH_STATIC_NULL { ((uint64_t)PMATH_TAGMASK_POINTER) << 32 }

Typedefs

- typedef int pmath_type_t

  The type or class of a pMath object.

- typedef int pmath_write_options_t

  Options for pmath_write().

- typedef void(∗ pmath_proc_t)(pmath_t)

  A simple procedure operating on an object.

- typedef void(∗ pmath_param_proc_t)(void ∗, pmath_t)

  A parameterized procedure operating on an object.

- typedef pmath_t(∗ pmath_func_t)(pmath_t)

  A simple function operating on an object and returning one.

- typedef unsigned int(∗ pmath_hash_func_t)(pmath_t)

  A hash function for an object.

- typedef pmath_bool_t(∗ pmath_equal_func_t)(pmath_t, pmath_t)

  A comparision function for two objects.

- typedef int(∗ pmath_compare_func_t)(pmath_t, pmath_t)

  A comparision function to determine the order of two objects.

Enumerations

- enum {

  PMATH_TYPE_SHIFT_MP_FLOAT = 0, PMATH_TYPE_SHIFT_MP_INT, PMATH_TYPE_SHIFT_QUOTIENT, PMATH_TYPE_SHIFT_BIGSTRING,

PMATH_TYPE_SHIFT_SYMBOL, PMATH_TYPE_SHIFT_EXPRESSION_GENERAL,

PMATH_TYPE_SHIFT_EXPRESSION_GENERAL_PART, PMATH_TYPE_SHIFT_RESERVED_1,

PMATH_TYPE_SHIFT_CUSTOM, PMATH_TYPE_SHIFT_COUNT }

- enum {

  PMATH_TYPE_MP_INT = 1 << PMATH_TYPE_SHIFT_MP_INT, PMATH_TYPE_QUOTIENT = 1 << PMATH_TYPE_SHIFT_QUOTIENT,

  PMATH_TYPE_MP_FLOAT = 1 << PMATH_TYPE_SHIFT_MP_FLOAT, PMATH_TYPE_BIGSTRING = 1 << PMATH_TYPE_SHIFT_BIGSTRING,

  PMATH_TYPE_SYMBOL = 1 << PMATH_TYPE_SHIFT_SYMBOL, PMATH_TYPE_EXPRESSION_GENERAL = 1 << PMATH_TYPE_SHIFT_EXPRESSION_GENERAL,

  PMATH_TYPE_EXPRESSION_GENERAL_PART = 1 << PMATH_TYPE_SHIFT_EXPRESSION_GENERAL_PART, PMATH_TYPE_EXPRESSION = PMATH_TYPE_EXPRESSION_GENERAL | PMATH_TYPE_EXPRESSION_GENERAL_PART,

  PMATH_TYPE_CUSTOM = 1 << PMATH_TYPE_SHIFT_CUSTOM }

- enum {

  PMATH_WRITE_OPTIONS_FULLEXPR = 1 << 0, PMATH_WRITE_OPTIONS_FULLSTR = 1 << 1,

  PMATH_WRITE_OPTIONS_FULLNAME = 1 << 2, PMATH_WRITE_OPTIONS_INPUTEXPR = 1 << 3 }

Functions

- pmath_t PMATH_FROM_TAG (uint32_t tag, int32_t value)
- pmath_t PMATH_FROM_INT32 (int32_t i)
- pmath_t PMATH_FROM_PTR (void ∗p)
- size_t pmath_object_bytecount (pmath_t obj)

  Get the byte count of an object.

- unsigned int pmath_hash (pmath_t obj)

  Calculates an object's hash value.

- int pmath_compare (pmath_t objA, pmath_t objB)

  Compares two objects syntactically.

- void pmath_write (pmath_t obj, pmath_write_options_t options, void(∗write)(void ∗user, const uint16_t ∗data, int len), void ∗user)

  Write an object to a stream.

- void pmath_write_ex (struct pmath_write_ex_t ∗info, pmath_t obj)

Advanced function to write an object to a stream.

- pmath_bool_t pmath_is_evaluated (pmath_t obj)

  Test whether an object is already evaluated.

- void pmath_write_with_pagewidth (pmath_t obj, pmath_write_options_t options, void(∗write)(void ∗user, const uint16_t ∗data, int len), void ∗user, int page_width, int indentation_width)

  Write an object to a stream with a maximum line width.

Variables

- static PMATH_UNUSED const pmath_t PMATH_UNDEFINED

  Magic value to indicate unset variable values/...

- static PMATH_UNUSED const pmath_t PMATH_NULL

  The NULL pointer. /\/ in pMath.

9.4.1 Detailed Description

The basic class for all pMath objects.

pMath works on objects. They can be expressions (trees of pMath objects), symbols, numbers, strings or ‘magic objects‘ (special integer values).

For efficiency reasons, 32 bit integers, double precision floating point values, short strings up to 2 characters and magic values are stored inline in the pmath_t struct. The struct size is only 8 bytes (= sizeof(double)) thanks to a technique called NaN-boxing.

This implementation could change in future version and/or on different architectures, so do not rely on it.

See also:

Object Utility Functions

9.4.2 Define Documentation

9.4.2.1 #define PMATH_ABORT_EXCEPTION PMATH_FROM_TAG(PMATH_TAG_MAGIC, 254)

9.4.2.2 #define PMATH_STATIC_NULL { ((uint64_t)PMATH_TAGMASK_POINTER) << 32 }

9.4.2.3 #define PMATH_STATIC_UNDEFINED { (((uint64_t)PMATH_TAG_MAGIC) << 32) | 255 }

9.4.2.4 #define PMATH_TAG_INT32 (PMATH_TAGMASK_NONDOUBLE | 0x20000)

9.4.2.5 #define PMATH_TAG_INVALID (PMATH_TAGMASK_NONDOUBLE | 0xFFFFF)

9.4.2.6 #define PMATH_TAG_MAGIC (PMATH_TAGMASK_NONDOUBLE | 0x10000)

9.4.2.7 #define PMATH_TAG_STR0 (PMATH_TAGMASK_NONDOUBLE | 0x30000)

9.4.2.8 #define PMATH_TAG_STR1 (PMATH_TAGMASK_NONDOUBLE | 0x40000)

9.4.2.9 #define PMATH_TAG_STR2 (PMATH_TAGMASK_NONDOUBLE | 0x50000)

9.4.2.10 #define PMATH_TAGMASK_BITCOUNT 12

9.4.2.11 #define PMATH_TAGMASK_NONDOUBLE 0x7FF00000U

9.4.2.12 #define PMATH_TAGMASK_POINTER 0xFFF00000U

9.4.2.13 #define PMATH_THREAD_KEY_PARSERARGUMENTS PMATH_FROM_TAG(PMATH_TAG_MAGIC, 253)

9.4.2.14 #define PMATH_THREAD_KEY_PARSESYMBOLS PMATH_FROM_TAG(PMATH_TAG_MAGIC, 252)

9.4.3 Typedef Documentation

9.4.3.1 typedef int(∗ pmath_compare_func_t)(pmath_t, pmath_t)

A comparision function to determine the order of two objects.

The return value is <0, =0 or >0, if the first argument is less, equal to or greater than the second respectively. Both arguments won't be destroyed by the function.

9.4.3.2 typedef pmath_bool_t(∗ pmath_equal_func_t)(pmath_t, pmath_t)

A comparision function for two objects.

The return value is nonzero, if both objects equal and zero otherwise. Note that a pmath_compare_func_t cannot be cast to pmath_equal_func_t, because their return values have opposite meanings.

9.4.3.3 typedef pmath_t(∗ pmath_func_t)(pmath_t)

A simple function operating on an object and returning one.

It depends on the context whether the argument is destroyed by the function or not.

9.4.3.4 typedef unsigned int(∗ pmath_hash_func_t)(pmath_t)

A hash function for an object.

If two objects equal, their hash values equal.

9.4.3.5 typedef void(* pmath_param_proc_t)(void *, pmath_t)

A parameterized procedure operating on an object.

It depends on the context whether the (second) argument is destroyed by the procedure or not.

9.4.3.6 typedef void(* pmath_proc_t)(pmath_t)

A simple procedure operating on an object.

It depends on the context whether the argument is destroyed by the procedure or not.

9.4.3.7 typedef int pmath_type_t

The type or class of a pMath object.

This is a bitset of the `PMATH_TYPE_XXX` constants:

- `PMATH_TYPE_MP_FLOAT`: The object is a floating point number with arbitrary precision. You can cast it to pmath_float_t and pmath_number_t.

- `PMATH_TYPE_INTEGER`: The object is an integer value. You can cast it to pmath_integer_t, pmath_rational_t and pmath_number_t.

- `PMATH_TYPE_QUOTIENT`: The object is a reduced quotient of two integer values, where the denominator is never 0 or 1. You can cast quotient objects to pmath_rational_t and thus to pmath_number_t too.

- `PMATH_TYPE_BIGSTRING`: The object is a string. You can cast it to pmath_string_t.

- `PMATH_TYPE_SYMBOL`: The object is a symbol. You can cast it to pmath_symbol_t.

- `PMATH_TYPE_CUSTOM`: The object is a custom object. You can cast it to pmath_custom_t.

9.4.3.8 typedef int pmath_write_options_t

Options for pmath_write().

These options can be one or more of the following:

- PMATH_WRITE_OPTIONS_FULLEXPR All expressions are written in the form f(a, b, ...) without any syntactic sugar.

Supersedes PMATH_WRITE_OPTIONS_INPUTEXPR.

- PMATH_WRITE_OPTIONS_FULLSTR Strings are written with quotes and escape sequences.

- PMATH_WRITE_OPTIONS_FULLNAME Names are written with their full namespace path.

- PMATH_WRITE_OPTIONS_INPUTEXPR Expressions are written in a form that is valid pMath input.

Note that this does not automatically imply PMATH_WRITE_OPTIONS_FULLSTR.

9.4.4 Enumeration Type Documentation

9.4.4.1 anonymous enum

Enumerator:

PMATH_TYPE_SHIFT_MP_FLOAT
PMATH_TYPE_SHIFT_MP_INT
PMATH_TYPE_SHIFT_QUOTIENT
PMATH_TYPE_SHIFT_BIGSTRING
PMATH_TYPE_SHIFT_SYMBOL
PMATH_TYPE_SHIFT_EXPRESSION_GENERAL
PMATH_TYPE_SHIFT_EXPRESSION_GENERAL_PART
PMATH_TYPE_SHIFT_RESERVED_1
PMATH_TYPE_SHIFT_CUSTOM
PMATH_TYPE_SHIFT_COUNT

9.4.4.2 anonymous enum

Enumerator:

PMATH_TYPE_MP_INT
PMATH_TYPE_QUOTIENT
PMATH_TYPE_MP_FLOAT
PMATH_TYPE_BIGSTRING

PMATH_TYPE_SYMBOL

PMATH_TYPE_EXPRESSION_GENERAL

PMATH_TYPE_EXPRESSION_GENERAL_PART

PMATH_TYPE_EXPRESSION

PMATH_TYPE_CUSTOM

9.4.4.3 anonymous enum

Enumerator:

PMATH_WRITE_OPTIONS_FULLEXPR

PMATH_WRITE_OPTIONS_FULLSTR

PMATH_WRITE_OPTIONS_FULLNAME

PMATH_WRITE_OPTIONS_INPUTEXPR

9.4.5 Function Documentation

9.4.5.1 int pmath_compare (pmath_t *objA*, pmath_t *objB*) `[inherited]`

Compares two objects syntactically.

Parameters:

objA The first object.

objB The second one.

Returns:

< 0 if *objA* is less than *objB*, == 0 if both are equal and > 0 if *objA* is greater than *objB*.

'syntactically' means that for two symbols X and Y, pmath_compare(X, Y) < 0 even if X:=2 and Y:=1, because X appears before Y in the alphabet.

Note:

pmath_equals(A, B) might return FALSE although pmath_compare(A, B) == 0 e.g. for an integer A and q floating point value B.

9.4.5.2 pmath_t PMATH_FROM_INT32 (int32_t *i*)

9.4.5.3 pmath_t PMATH_FROM_PTR (void ∗ *p*)

9.4.5.4 pmath_t PMATH_FROM_TAG (uint32_t *tag*, int32_t *value*)

9.4.5.5 unsigned int pmath_hash (pmath_t *obj*) `[inherited]`

Calculates an object's hash value.

Parameters:

obj The object.

Returns:

A hash value.

pmath_equals(A, B) implies pmath_hash(A) == pmath_hash(B).

9.4.5.6 pmath_bool_t pmath_is_evaluated (pmath_t *obj*) `[inherited]`

Test whether an object is already evaluated.

Parameters:

obj Any pMath object. It will ∗not∗ be freed.

Returns:

TRUE if a call to pmath_evaluate would not change the object.

9.4.5.7 size_t pmath_object_bytecount (pmath_t *obj*)

Get the byte count of an object.

Parameters:

obj The object. It wont be freed

Returns:

An estimate for the memory usage of this object. Symbols count as 0. Any elements that reference to the same object are treated as distinct.

9.4.5.8 void pmath_write (pmath_t *obj*, pmath_write_options_t *options*, void(∗)(void ∗user, const uint16_t ∗data, int len) *write*, void ∗ *user*) `[inherited]`

Write an object to a stream.

Parameters:

- ***obj*** The object to be written.
- ***options*** Some options defining the format.
- ***write*** The stream's output function.
- ***user*** The user-argument of write (e.g. the stream itself).

See also:

pmath_utf8_writer

9.4.5.9 void pmath_write_ex (struct pmath_write_ex_t ∗ *info*, pmath_t *obj*) `[inherited]`

Advanced function to write an object to a stream.

Parameters:

- ***info*** All the acutal parameters.
- ***obj*** The object to be written.

See also:

pmath_write

9.4.5.10 void pmath_write_with_pagewidth (pmath_t *obj*, pmath_write_options_t *options*, void(∗)(void ∗user, const uint16_t ∗data, int len) *write*, void ∗ *user*, int *page_width*, int *indentation_width*) `[inherited]`

Write an object to a stream with a maximum line width.

Parameters:

- ***obj*** The object to be written.
- ***options*** Some options defining the format.

write The stream's output function.

user The user-argument of write (e.g. the stream itself).

page_width The page width. This should be at least 6.

indentation_width The minimum number of spaces to insert after every implicit line break.

If *page_width* <0, the global variable $PageWidth is used. Line breaks will generally not appear within single tokens (e.g. very long symbol names) when those appear inside `InputForm` or when *options* contains `PMATH_WRITE_OPTIONS_INPUTEXPR`.

See also:

pmath_write

9.4.6 Variable Documentation

9.4.6.1 PMATH_UNUSED const pmath_t PMATH_NULL `[static]`

The NULL pointer. /\/ in pMath.

9.4.6.2 PMATH_UNUSED const pmath_t PMATH_UNDEFINED `[static]`

Magic value to indicate unset variable values/...

9.5 Strings

String objects in pMath.

Data Structures

- struct pmath_cstr_writer_info_t

  Additional information for pmath_utf8_writer() or pmath_native_writer().

- class pmath_string_t

  The string class.

Defines

- #define PMATH_C_STRING(cstr) pmath_string_insert_latin1(PMATH_NULL, 0, (cstr), -1)

Short form to convert a C String to a pMath String.

- PMATH_C_STRING(cstr)

  Short form to convert a C String to a pMath String.

Typedefs

- typedef pmath_t pmath_string_t

Functions

- void pmath_utf8_writer (void ∗user, const uint16_t ∗data, int len)

  A write *function for pmath_write() that converts to utf8.*

- void pmath_native_writer (void ∗user, const uint16_t ∗data, int len)

  A write *function for pmath_write() that converts to the current console encosing.*

- uint32_t pmath_char_from_name (const char ∗name)

  Get a named character.

- const char ∗ pmath_char_to_name (uint32_t unichar)

  Get a character's name.

- const uint16_t ∗ pmath_char_parse (const uint16_t ∗str, int maxlen, uint32_t ∗result)

  Parse an escaped character to a unicode codepoint.

- pmath_string_t pmath_string_new (int capacity)

  Create an empty pMath String.

- pmath_string_t pmath_string_insert_latin1 (pmath_string_t str, int inspos, const char ∗ins, int inslen)

  Insert an Latin-1 encoded buffer into a pMath String.

- pmath_string_t pmath_string_from_utf8 (const char ∗str, int len)

  Convert an UTF-8 encoded buffer to a pMath String.

- char ∗ pmath_string_to_utf8 (pmath_string_t str, int ∗result_len)

  Convert a pMath string to a zero-terminated UTF-8 string.

- pmath_string_t pmath_string_from_native (const char ∗str, int len)

  Convert a string buffer in the current console character encoding to a pMath String.

- char ∗ pmath_string_to_native (pmath_string_t str, int ∗result_len)

  Convert a pMath string to a string in the current console character encoding.

- pmath_string_t pmath_string_insert_codepage (pmath_string_t str, int inspos, const char ∗ins, int inslen, const uint16_t ∗cp)

  Insert a byte string into a pMath string using a translation array.

- pmath_string_t pmath_string_insert_ucs2 (pmath_string_t str, int inspos, const uint16_t ∗ins, int inslen)

  Insert a UCS-2 buffer into a pMath String.

- pmath_string_t pmath_string_insert (pmath_string_t str, int inspos, pmath_string_t ins)

  Insert one pMath String into another pMath String.

- pmath_string_t pmath_string_concat (pmath_string_t prefix, pmath_string_t postfix)

  Concatenate two pMath Strings.

- pmath_string_t pmath_string_part (pmath_string_t string, int start, int length)

  Extract a substring of a pMath String.

- const uint16_t ∗ pmath_string_buffer (pmath_string_t ∗string)

  Get a string's buffer for reading.

- int pmath_string_length (pmath_string_t string)

  Get a string's length.

- pmath_bool_t pmath_string_equals_latin1 (pmath_string_t string, const char ∗latin1)

  Compare a pMath string with a C string.

9.5.1 Detailed Description

String objects in pMath.

pMath stores strings in UCS-2 format (like Java and Windows NT). But pMath strings are not zero terminated.

Do not confuse pMath String characters (uint16_t) with wchar_t: sizeof(wchar_t) differs on different Systems (Linux: 4 bytes, Windows: 2 bytes). So you cannot simply convert wchar_t∗ strings to pMath strings.

9.5.2 Define Documentation

9.5.2.1 PMATH_C_STRING(cstr) `[inherited]`

Short form to convert a C String to a pMath String.

Parameters:

cstr A C String (zero-terminated char buffer).

Returns:

A pMath String representing the Latin-1 C string *cstr*.

This is a wrapper macro around pmath_string_insert_latin1().

9.5.2.2 #define PMATH_C_STRING(cstr) pmath_string_insert_latin1(PMATH_NULL, 0, (cstr), -1)

Short form to convert a C String to a pMath String.

Parameters:

cstr A C String (zero-terminated char buffer).

Returns:

A pMath String representing the Latin-1 C string *cstr*.

This is a wrapper macro around pmath_string_insert_latin1().

9.5.3 Typedef Documentation

9.5.3.1 typedef pmath_t pmath_string_t

9.5.4 Function Documentation

9.5.4.1 uint32_t pmath_char_from_name (const char ∗ *name*)

Get a named character.

Parameters:

name The ASCII-name of the character. e.g. "Sum"

Returns:

The character code or 0xFFFFFFFFU on error

9.5.4.2 const uint16_t∗ pmath_char_parse (const uint16_t ∗ *str*, int *maxlen*, uint32_t ∗ *result*)

Parse an escaped character to a unicode codepoint.

Parameters:

str A string of the form \[name] or or or ...

maxlen The buffer length of *str*.

result Here goes the parsed character, 0xFFFFFFFFU on error.

Returns:

The end of the parsed character or the error position.

9.5.4.3 const char∗ pmath_char_to_name (uint32_t *unichar*)

Get a character's name.

Parameters:

unichar A unicode character

Returns:

The ASCII-name or NULL if it is unnamed

9.5.4.4 void pmath_native_writer (void ∗ *user*, const uint16_t ∗ *data*, int *len*)

A *write* function for pmath_write() that converts to the current console encosing.

This callback function is used like pmath_utf8_writer().

9.5.4.5 const uint16_t ∗ pmath_string_buffer (pmath_string_t ∗ *string*) [inherited]

Get a string's buffer for reading.

Parameters:

string A pointer to a string.

Returns:

A pointer to the string's buffer. This buffer is guaranteed to be pmath_string_-length(str) $*$ sizeof(uint16_t) bytes long.

Do not forget that pMath strings are not zero-terminated.

9.5.4.6 pmath_string_t pmath_string_concat (pmath_string_t *prefix*, pmath_string_t *postfix*) `[inherited]`

Concatenate two pMath Strings.

Parameters:

prefix A pMath String. It will be freed.

postfix A pMath String. It will be freed.

Returns:

PMATH_NULL on failure or a pMath String that consists of `prefix` followed by `postfix`. You must destroy it.

If one of the two strings is PMATH_NULL, the other string will be returned.

9.5.4.7 pmath_bool_t pmath_string_equals_latin1 (pmath_string_t *string*, const char $*$ *latin1*) `[inherited]`

Compare a pMath string with a C string.

Parameters:

string A string. It wont be freed.

latin1 A C string (zero terminated).

Returns:

Whether the two string are equals.

This function is a short form for

```
tmp = PMATH_C_STRING(latin1);
result = pmath_equals(string, tmp);
pmath_unref(tmp);
```

9.5.4.8 pmath_string_t pmath_string_from_native (const char * *str*, int *len*) [inherited]

Convert a string buffer in the current console character encoding to a pMath String.

Parameters:

str A byte string. It wont be freed.

len The byte-length of `ins` or -1 if it is zero-terminated.

Returns:

PMATH_NULL on failure or a pMath String. You must destroy it.

9.5.4.9 pmath_string_t pmath_string_from_utf8 (const char * *str*, int *len*) [inherited]

Convert an UTF-8 encoded buffer to a pMath String.

Parameters:

str A byte string. It wont be freed.

len The byte-length of `ins` or -1 if it is zero-terminated.

Returns:

PMATH_NULL on failure or a pMath String. You must destroy it.

9.5.4.10 pmath_string_t pmath_string_insert (pmath_string_t *str*, int *inspos*, pmath_string_t *ins*) [inherited]

Insert one pMath String into another pMath String.

Parameters:

str A pMath String or PMATH_NULL. It will be freed.

inspos The position, at which `ins` should be inserted in `str`.

ins A pMath String or PMATH_NULL. It will be freed.

Returns:

PMATH_NULL on failure or a pMath String. You must destroy it.

9.5.4.11 pmath_string_t pmath_string_insert_codepage (pmath_string_t *str*, int *inspos*, const char ∗ *ins*, int *inslen*, const uint16_t ∗ *cp*) `[inherited]`

Insert a byte string into a pMath string using a translation array.

Parameters:

str A pMath String or PMATH_NULL. It will be freed.

inspos The position, at which `ins` should be inserted in `str`.

ins A byte string.

inslen The length of `ins` or -1 if it is zero-terminated.

cp An array of 256 uint16_t values that are used to convert bytes to UCS-2 characters.

Returns:

PMATH_NULL on failure or a pMath String. You must destroy it.

If `str` is PMATH_NULL, it is assumed to be the empty string.

9.5.4.12 pmath_string_t pmath_string_insert_latin1 (pmath_string_t *str*, int *inspos*, const char ∗ *ins*, int *inslen*) `[inherited]`

Insert an Latin-1 encoded buffer into a pMath String.

Parameters:

str A pMath String or PMATH_NULL. It will be freed.

inspos The position, at which `ins` should be inserted in `str`.

ins A byte string.

inslen The length of `ins` or -1 if it is zero-terminated.

Returns:

PMATH_NULL on failure or a pMath String. You must destroy it.

If `str` is PMATH_NULL, it is assumed to be the empty string. The result is equivalent to a call to pmath_string_insert_codepage() with a codepage that translates every byte **b** to (uint16_t)(unsigned char)b.

9.5.4.13 pmath_string_t pmath_string_insert_ucs2 (pmath_string_t *str*, int *inspos*, const uint16_t ∗ *ins*, int *inslen*) `[inherited]`

Insert a UCS-2 buffer into a pMath String.

Parameters:

str A pMath String or PMATH_NULL. It will be freed.

inspos The position, at which `ins` should be inserted in `str`.

ins A uint16_t string. This is *not* a wchar_t string.

inslen The length of `ins` or -1 if it is zero-terminated.

Returns:

PMATH_NULL on failure or a pMath String. You must destroy it.

If `str` is PMATH_NULL, it is assumed to be the empty string.

9.5.4.14 int pmath_string_length (pmath_string_t *string*) `[inherited]`

Get a string's length.

Parameters:

string A string. It remains valid after the function call, so you have to destroy it manually.

Returns:

The length (in uint16_t characters) of the string. It is never negative.

9.5.4.15 pmath_string_t pmath_string_new (int *capacity*) `[inherited]`

Create an empty pMath String.

Parameters:

capacity The initial capacity of the string. Must not be negative.

Returns:

A new pMath String or PMATH_NULL on failure. You must destroy it.

9.5.4.16 pmath_string_t pmath_string_part (pmath_string_t *string*, int *start*, int *length*) `[inherited]`

Extract a substring of a pMath String.

Parameters:

string A pMath String. It will be freed.

start the substring's start index.

length the substring's length or -1 for the whole substring beginng at start.

Returns:

PMATH_NULL on failure or a pMath String.

If start or start+length are out of bounds, thy will be truncated, the the resulting string's length is not neccesaryly `length`.

9.5.4.17 char ∗ pmath_string_to_native (pmath_string_t *str*, int ∗ *result_len*) `[inherited]`

Convert a pMath string to a string in the current console character encoding.

Parameters:

str A pMath string. It wont be freed.

result_len Position, where the string length of the returned buffer may be stored.

Returns:

A zero terminated string or PMATH_NULL on error. You have to free the memory with pmath_mem_free(result, ∗size_ptr).

Note:

pMath strings may contain embedded '\0', but C strings may not. However, the conversion is done to the whole string even though your C functions will only *see* the content up to the first '\0'.

9.5.4.18 char ∗ pmath_string_to_utf8 (pmath_string_t *str*, int ∗ *result_len*) `[inherited]`

Convert a pMath string to a zero-terminated UTF-8 string.

Parameters:

str A pMath string. It wont be freed.

result_len Position, where the string length of the returned buffer may be stored.

Returns:

A zero-terminated UTF-8 string or PMATH_NULL on error. You have to free the memory with pmath_mem_free(result, ∗size_ptr).

Note:

> pMath strings may contain embedded '\0', but C strings may not. However, the conversion is done to the whole string even though your C functions will only *see* the content up to the first '\0'.

9.5.4.19 void pmath_utf8_writer (void * *user*, const uint16_t * *data*, int *len*)

A *write* function for pmath_write() that converts to utf8.

pmath_write() writes output as utf16/ucs2. This function can be used to convert to utf8 on the fly. The *user* parameter to pmath_write must point to a pmath_cstr_writer_info_t.

Here is an example on how to use it:

```
void my_utf8_output(FILE *f, const char *str){
  fprintf(f, "%s", str);
}

...

pmath_cstr_writer_info_t info;
info.write_cstr = (void(*)(void*,const char*))my_utf8_output;
info.user = stdout; // will be first argument of my_utf8_output

pmath_print(some_object, some_options, pmath_utf8_writer, &info);
```

9.6 Symbols

Symbol objects in pMath.

Data Structures

- class pmath_symbol_t

  The Symbol class.

Typedefs

- typedef pmath_t pmath_symbol_t
- typedef int pmath_symbol_attributes_t

  The (bitset) type of symbol attributes.

Enumerations

- enum {
  PMATH_SYMBOL_ATTRIBUTE_PROTECTED = 1 << 0, PMATH_SYMBOL_ATTRIBUTE_HOLDFIRST = 1 << 1,

  PMATH_SYMBOL_ATTRIBUTE_HOLDREST = 1 << 2, PMATH_SYMBOL_ATTRIBUTE_HOLDALL = PMATH_SYMBOL_ATTRIBUTE_HOLDFIRST | PMATH_SYMBOL_ATTRIBUTE_HOLDREST,

  PMATH_SYMBOL_ATTRIBUTE_SYMMETRIC = 1 << 3, PMATH_SYMBOL_ATTRIBUTE_ASSOCIATIVE = 1 << 4,

  PMATH_SYMBOL_ATTRIBUTE_NHOLDFIRST = 1 << 5, PMATH_SYMBOL_ATTRIBUTE_NHOLDREST = 1 << 6,

  PMATH_SYMBOL_ATTRIBUTE_NHOLDALL = PMATH_SYMBOL_ATTRIBUTE_NHOLDFIRST | PMATH_SYMBOL_ATTRIBUTE_NHOLDREST, PMATH_SYMBOL_ATTRIBUTE_TEMPORARY = 1 << 7,

  PMATH_SYMBOL_ATTRIBUTE_LISTABLE = 1 << 8, PMATH_SYMBOL_ATTRIBUTE_DEEPHOLDALL = 1 << 9,

  PMATH_SYMBOL_ATTRIBUTE_HOLDALLCOMPLETE = 1 << 10, PMATH_SYMBOL_ATTRIBUTE_ONEIDENTITY = 1 << 11,

  PMATH_SYMBOL_ATTRIBUTE_THREADLOCAL = 1 << 12, PMATH_SYMBOL_ATTRIBUTE_NUMERICFUNCTION = 1 << 13,

  PMATH_SYMBOL_ATTRIBUTE_READPROTECTED = 1 << 14, PMATH_SYMBOL_ATTRIBUTE_SEQUENCEHOLD = 1 << 15,

  PMATH_SYMBOL_ATTRIBUTE_REMOVED = 1 << 16, PMATH_SYMBOL_ATTRIBUTE_DEFINITEFUNCTION = 1 << 17 }

Functions

- pmath_symbol_t pmath_symbol_get (pmath_string_t name, pmath_bool_t create)

  Get a symbol by its fully qualified name.

- pmath_symbol_t pmath_symbol_create_temporary (pmath_string_t name, pmath_bool_t unique)

  Create a new temporary symbol.

- pmath_symbol_t pmath_symbol_find (pmath_string_t name, pmath_bool_t create)

  Find a symbol in the current namespace search path.

- pmath_string_t pmath_symbol_name (pmath_symbol_t symbol)

  Get a symbol's name.

- pmath_symbol_attributes_t pmath_symbol_get_attributes (pmath_symbol_t symbol)

  Get a symbol's attributes.

- void pmath_symbol_set_attributes (pmath_symbol_t symbol, pmath_symbol_attributes_t attr)

  Set a symbol's attributes.

- pmath_t pmath_symbol_get_value (pmath_symbol_t symbol)

  Get a symbol's value.

- void pmath_symbol_set_value (pmath_symbol_t symbol, pmath_t value)

  Set a symbol's value.

- void pmath_symbol_synchronized (pmath_symbol_t symbol, pmath_callback_t callback, void ∗data)

  Execute a function synchronized to a symbol.

- void pmath_symbol_update (pmath_symbol_t symbol)

  Update a symbol manually.

- void pmath_symbol_remove (pmath_symbol_t symbol)

  Remove a symbol completely from the system.

- pmath_symbol_t pmath_symbol_iter_next (pmath_symbol_t old)

  Iterate through the global symbol table.

9.6.1 Detailed Description

Symbol objects in pMath.

9.6.2 Typedef Documentation

9.6.2.1 typedef int pmath_symbol_attributes_t

The (bitset) type of symbol attributes.

A pMath symbol (here called ‘sym‘) can have one or more of the following values (concatenated with "|"):

- `PMATH_SYMBOL_ATTRIBUTE_PROTECTED`

  Any assignment to sym will fail.

- PMATH_SYMBOL_ATTRIBUTE_HOLDFIRST

  When evaluating ‘sym(a,b,...)‘, the first argument (a) will not be evaluated automatically.

- PMATH_SYMBOL_ATTRIBUTE_HOLDREST

  When evaluating ‘sym(a,b,...)‘ all the arguments b,... will not be evaluated automatically.

- PMATH_SYMBOL_ATTRIBUTE_HOLDALL

  combines HOLDFIRST and HOLDREST.

- PMATH_SYMBOL_ATTRIBUTE_SYMMETRIC

  An expression ‘sym(a,b,...)‘ will be sorted automatically and thus sym(a,b) = sym(b,a).

- PMATH_SYMBOL_ATTRIBUTE_ASSOCIATIVE

  An expression ‘sym(...,sym(a,...,z),...)‘ will be flattened automatically to sym(...,a,...,z,...).

- PMATH_SYMBOL_ATTRIBUTE_NHOLDFIRST

  The first argument (a) of ‘sym(a,b,...)‘ will not be affected by Approximate(sym(a,b,...)).

- PMATH_SYMBOL_ATTRIBUTE_NHOLDREST

  All the argument ‘b,...‘ in ‘sym(a,b,...)‘ will not be affected by Approximate(sym(a,b,...)).

- PMATH_SYMBOL_ATTRIBUTE_NHOLDALL

  combines NHOLDFIRST and NHOLDREST.

- PMATH_SYMBOL_ATTRIBUTE_TEMPORARY

  The symbol sym will be deleted immediately when it is no longer referenced. It will be freed automatically (but not immediately), when there is no external reference to the symbol (just the symbol's own function definitions/...).

- PMATH_SYMBOL_ATTRIBUTE_LISTABLE

  Any expression sym(...) will be threaded automatically over lists. (e.g. {a,b} + {c,d} becomes {a+c, b+d}).

- PMATH_SYMBOL_ATTRIBUTE_DEEPHOLDALL The arguments ‘a,...‘ in an expression ‘sym(...)(a,...)‘ will not be evaluated automatically.

- PMATH_SYMBOL_ATTRIBUTE_HOLDALLCOMPLETE

  Like HOLDALL, but in an expression ‘sym(a,b,...)‘ all arguments (a,b,...) wont be touched even if they have the form ‘eval(...)‘. Additionally, rules for sym defined in one of its arguments (e.g. ‘a: sym(a):= "hi"‘) wont be used.

- PMATH_SYMBOL_ATTRIBUTE_ONEIDENTITY

  Used for pattern matching (in combination with ASSOCIATIVE) to say that ‘sym(x)‘ matches x. Note that it does not automatically evaluate ‘sym(x)‘ to x.

- `PMATH_SYMBOL_ATTRIBUTE_THREADLOCAL`

  The symbol's value is local to the current thread. That means, an assignment to sym in one thread wont affect it in another thread.

- `PMATH_SYMBOL_ATTRIBUTE_NUMERICFUNCTION`

  'sym(x,...)' is numeric if all the arguments are numeric.

- `PMATH_SYMBOL_ATTRIBUTE_READPROTECTED` '??sym' wont print out the value/function definitions for sym.

- `PMATH_SYMBOL_ATTRIBUTE_SEQUENCEHOLD` Sequence(...) wont be sliced when it appears as an argument to 'sym(...)'

- `PMATH_SYMBOL_ATTRIBUTE_REMOVED` The symbol was removed, but there are pending references to it.

- `PMATH_SYMBOL_ATTRIBUTE_DEFINITEFUNCTION` 'sym(ConditionalExpression(arg, cond))' becomes 'ConditionalExpression(sym(arg), cond)'

9.6.2.2 typedef pmath_t pmath_symbol_t

9.6.3 Enumeration Type Documentation

9.6.3.1 anonymous enum

Enumerator:

PMATH_SYMBOL_ATTRIBUTE_PROTECTED

PMATH_SYMBOL_ATTRIBUTE_HOLDFIRST

PMATH_SYMBOL_ATTRIBUTE_HOLDREST

PMATH_SYMBOL_ATTRIBUTE_HOLDALL

PMATH_SYMBOL_ATTRIBUTE_SYMMETRIC

PMATH_SYMBOL_ATTRIBUTE_ASSOCIATIVE

PMATH_SYMBOL_ATTRIBUTE_NHOLDFIRST

PMATH_SYMBOL_ATTRIBUTE_NHOLDREST

PMATH_SYMBOL_ATTRIBUTE_NHOLDALL

PMATH_SYMBOL_ATTRIBUTE_TEMPORARY

PMATH_SYMBOL_ATTRIBUTE_LISTABLE

PMATH_SYMBOL_ATTRIBUTE_DEEPHOLDALL

PMATH_SYMBOL_ATTRIBUTE_HOLDALLCOMPLETE

PMATH_SYMBOL_ATTRIBUTE_ONEIDENTITY
PMATH_SYMBOL_ATTRIBUTE_THREADLOCAL
PMATH_SYMBOL_ATTRIBUTE_NUMERICFUNCTION
PMATH_SYMBOL_ATTRIBUTE_READPROTECTED
PMATH_SYMBOL_ATTRIBUTE_SEQUENCEHOLD
PMATH_SYMBOL_ATTRIBUTE_REMOVED
PMATH_SYMBOL_ATTRIBUTE_DEFINITEFUNCTION

9.6.4 Function Documentation

9.6.4.1 pmath_symbol_t pmath_symbol_create_temporary (pmath_string_t *name*, pmath_bool_t *unique*) `[inherited]`

Create a new temporary symbol.

Parameters:

name The base name of the temporary symbol. It will be freed.
unique Whether to add a unique number to the symbol name.

Returns:

A new pMath Symbol. You must destroy it with pmath_unref(). It has the Temporary attribute.

the name of the returned symbol is of the form name$nnn (or name$ if unique is false)

If name already has the form "sym$nnn" or "sym$", the function acts as if name would be simply "sym".

If there already exists a symbol with the generated name, that symbol will be returned and its attributes will be set to Temporary before.

9.6.4.2 pmath_symbol_t pmath_symbol_find (pmath_string_t *name*, pmath_bool_t *create*) `[inherited]`

Find a symbol in the current namespace search path.

Parameters:

name The symbol's name. It will be freed.
create Whether to create a new symbol, if none was found.

Returns:

PMATH_NULL or a symbol called `name` that must be destroyed with pmath_unref().

9.6.4.3 pmath_symbol_t pmath_symbol_get (pmath_string_t *name*, pmath_bool_t *create*) `[inherited]`

Get a symbol by its fully qualified name.

Parameters:

name The symbol's name including its namespace. It will be freed.

create Whether to create a new symbol, if none was found.

Returns:

PMATH_NULL or a symbol called `name` that must be destroyed with pmath_unref().

9.6.4.4 pmath_symbol_attributes_t pmath_symbol_get_attributes (pmath_symbol_t *symbol*) `[inherited]`

Get a symbol's attributes.

Parameters:

symbol A pMath symbol. It wont be freed.

Returns:

The symbol's attributes.

9.6.4.5 pmath_t pmath_symbol_get_value (pmath_symbol_t *symbol*) `[inherited]`

Get a symbol's value.

Deprecated

Parameters:

symbol A pMath symbol.

Returns:

The symbol's value. You must free it with pmath_unref(). Note that not every object is evaluatable (e.g. Custom Objects).

9.6.4.6 pmath_symbol_t pmath_symbol_iter_next (pmath_symbol_t *old*) [inherited]

Iterate through the global symbol table.

Parameters:

old The previous symbol. It will be freed.

Returns:

The next symbol.

To actually iterate through the whole list, use the following pattern:

```
pmath_symbol_t iter = pmath_ref(PMATH_SYMBOL_LIST);
do{

  ... loop body here ...

  iter = pmath_symbol_iter_next(iter);
}while(iter && !pmath_same(iter, PMATH_SYMBOL_LIST));
pmath_unref(iter);
```

9.6.4.7 pmath_string_t pmath_symbol_name (pmath_symbol_t *symbol*) [inherited]

Get a symbol's name.

Parameters:

symbol A pMath symbol.

Returns:

The name of the symbol. You must destroy it with pmath_unref().

9.6.4.8 void pmath_symbol_remove (pmath_symbol_t *symbol*) [inherited]

Remove a symbol completely from the system.

Parameters:

symbol a pMath symbol. It will be freed.

Symbols with attribute protected wont be removed.

This function walks through the internal list of all known symbols and replaces any occurencies with ‘Symbol("name")‘.

There might be more references (e.g. on the stack or in other thread’s local variable tables), so it is possible that the symbol still exists in the system.

Note that all builtin symbols (the PMATH_SYMBOL_XXX) are also referenced in a seperate list and so cannot be removed completely from the system. However, their appearences in all other places will be removed. So this is a very dangerous function.

9.6.4.9 void pmath_symbol_set_attributes (pmath_symbol_t *symbol*, pmath_symbol_attributes_t *attr*) `[inherited]`

Set a symbol’s attributes.

Parameters:

symbol A pMath symbol. It wont be freed.

attr The new attributes.

9.6.4.10 void pmath_symbol_set_value (pmath_symbol_t *symbol*, pmath_t *value*) `[inherited]`

Set a symbol’s value.

Deprecated

Parameters:

symbol A pMath symbol. It wont be freed

value The new value. It will be freed.

This function ignores the Protected-attribute. You should only use it during symbol initialization and/or when you want to store non-evaluatable values in a symbol. In all other cases, evaluate an expression with the head PMATH_SYMBOL_ASSIGN or PMATH_SYMBOL_ASSIGNDELAYED.

9.6.4.11 void pmath_symbol_synchronized (pmath_symbol_t *symbol*, pmath_callback_t *callback*, void ∗ *data*) `[inherited]`

Execute a function synchronized to a symbol.

Deprecated

Parameters:

- ***symbol*** The symbol to lock. It wont be freed.
- ***callback*** The function to be executed when the symbol is locked.
- ***data*** A pointer that will be passed to callback.

See also:

Multithreading with pMath

9.6.4.12 void pmath_symbol_update (pmath_symbol_t *symbol*) `[inherited]`

Update a symbol manually.

Parameters:

- ***symbol*** A pMath symbol. It wont be freed.

You normally do not have to call this, since every change in a symbol yields an update. But there are some situations where you might to update it manually. The update mechanism is an optimization. Any expresseion or symbol, that is up to date while evaluation, wont be evaluated again. After an evaluation, expressions are updated automatically.

9.7 C++ Binding

Data Structures

- class Expr

  A wrapper for pmath_t and drived types.

- class String

  A wrapper for pmath_string_t.

- class Gather

  Utility class for emitting and gathering expressions/building lists.

- class File

  A wrapper for pMath file objects (data streams).

- class BinaryFile

A wrapper for pMath binary file objects (byte data streams).

- class ReadableBinaryFile

  A wrapper for readable pMath binary file objects (byte data streams).

- class WriteableBinaryFile

  A wrapper for writeable pMath binary file objects (byte data streams).

- class TextFile

  A wrapper for pMath text file objects (byte data streams).

- class ReadableTextFile

  A wrapper for pMath readable text file objects (byte data streams).

- class WriteableTextFile

  A wrapper for pMath writeable text file objects (byte data streams).

- class UserStream

  Abstract base class for C++ callbacks used as pMath files.

- class BinaryUserStream

  Abstract base class for C++ callbacks used as pMath binary files.

- class TextUserStream

  Abstract base class for C++ callbacks used as pMath text files.

Namespaces

- namespace pmath

  Provides the C++ binding.

9.7.1 Detailed Description

There exists a thin layer to easily use pMath with C++. This is usably prefreable over the C API because it handles reference counting/type checking automatically and leads to less "boilerplate code".

To use it, simply `#include <pmath-cpp.h>`. The classes are in the `pmath` namespace.

This namespace also containts numerous helper functions to easily construct expression trees. None of the classes and functions generate C++ exceptions, they are all fault tolerant (in contrast to most of the plain C API).

9.8 Parsing Code

Translating pMath code or boxes to pMath objects.

Data Structures

- class pmath_span_array_t

  Internal flat representaion of spans.

- class pmath_span_t

  Represents a span in a span-array.

Defines

- #define PMATH_RUN(code)

  Execute some pMath code.

- #define PMATH_RUN_ARGS(code, format,...)

  Execute some pMath code with arguments.

- #define PMATH_CHAR_INVISIBLECALL 0x2061

  The Function application character.

- #define PMATH_CHAR_VECTOR 0x21C0

  The arrow above names to indicate a vector.

- #define PMATH_CHAR_RULE 0x2192

  The " → " operator.

- #define PMATH_CHAR_RULEDELAYED 0x29F4

  The ":>" operator.

- #define PMATH_CHAR_ASSIGN 0x2254

  The ":=" operator.

- #define PMATH_CHAR_ASSIGNDELAYED 0x2A74

  The "::=" operator.

- #define PMATH_CHAR_INTEGRAL_D 0x2146

  The integral "d".

- #define PMATH_CHAR_PIECEWISE 0xF361

  The left curly bracket for cases.

- #define PMATH_CHAR_ALIASDELIMITER 0xF764

The character inserted by Richmath with ESCAPE or CAPSLOCK.

- #define PMATH_CHAR_ALIASINDICATOR 0xF768

  A character that looks like PMATH_CHAR_ALIASDELIMITER but has no effect.

- #define PMATH_CHAR_LEFT_BOX 0xFFF9

  Start of box code inside a string.

- #define PMATH_CHAR_RIGHT_BOX 0xFFFB

  End of box code inside a string.

- #define PMATH_CHAR_BOX 0xFDD0

  Represents a box.

- #define PMATH_CHAR_PLACEHOLDER 0xFFFD

  The Placeholder character. In richmath, type CAPSLOCK pl CAPSLOCK to insert it.

Typedefs

- typedef struct _pmath_span_array_t pmath_span_array_t
- typedef struct _pmath_span_t pmath_span_t

Enumerations

- enum pmath_token_t {
  PMATH_TOK_NONE, PMATH_TOK_SPACE,
  PMATH_TOK_DIGIT, PMATH_TOK_STRING,
  PMATH_TOK_NAME, PMATH_TOK_NAME2,
  PMATH_TOK_BINARY_LEFT, PMATH_TOK_BINARY_RIGHT,
  PMATH_TOK_BINARY_LEFT_AUTOARG, PMATH_TOK_BINARY_LEFT_OR_PREFIX,
  PMATH_TOK_NARY, PMATH_TOK_NARY_AUTOARG,
  PMATH_TOK_NARY_OR_PREFIX, PMATH_TOK_POSTFIX_OR_PREFIX,
  PMATH_TOK_PREFIX, PMATH_TOK_POSTFIX,
  PMATH_TOK_CALL, PMATH_TOK_LEFTCALL,
  PMATH_TOK_LEFT, PMATH_TOK_RIGHT,
  PMATH_TOK_PRETEXT, PMATH_TOK_ASSIGNTAG,
  PMATH_TOK_PLUSPLUS, PMATH_TOK_COLON,
  PMATH_TOK_TILDES, PMATH_TOK_SLOT,
  PMATH_TOK_QUESTION, PMATH_TOK_INTEGRAL,
  PMATH_TOK_COMMENTEND }

Token classes known in the pMath language.

- enum {
PMATH_PREC_ANY = 0, PMATH_PREC_SEQ = 10,
PMATH_PREC_EVAL = 20, PMATH_PREC_ASS = 30,
PMATH_PREC_MODY = 40, PMATH_PREC_LAZY = 50,
PMATH_PREC_FUNC = 60, PMATH_PREC_REPL = 80,
PMATH_PREC_RULE = 90, PMATH_PREC_MAP = 100,
PMATH_PREC_STR = 110, PMATH_PREC_COND = 120,
PMATH_PREC_ALT = 130, PMATH_PREC_OR = 150,
PMATH_PREC_XOR = 155, PMATH_PREC_AND = 160,
PMATH_PREC_ARROW = 170, PMATH_PREC_REL = 180,
PMATH_PREC_UNION = 190, PMATH_PREC_ISECT = 200,
PMATH_PREC_RANGE = 210, PMATH_PREC_ADD = 220,
PMATH_PREC_CIRCADD = 230, PMATH_PREC_PLUMI = 240,
PMATH_PREC_CIRCMUL = 250, PMATH_PREC_MUL = 260,
PMATH_PREC_DIV = 270, PMATH_PREC_MIDDOT = 280,
PMATH_PREC_CROSS = 290, PMATH_PREC_MUL2 = 300,
PMATH_PREC_POW = 310, PMATH_PREC_FAC = 320,
PMATH_PREC_APL = 330, PMATH_PREC_REPEAT = 340,
PMATH_PREC_TEST = 350, PMATH_PREC_INC = 360,
PMATH_PREC_CALL = 400, PMATH_PREC_DIFF = 410,
PMATH_PREC_PRIM = 1000 }

Functions

- pmath_span_array_t ∗ pmath_spans_from_string (pmath_string_t ∗code, pmath_string_t(∗line_reader)(void ∗), pmath_bool_t(∗subsuperscriptbox_at_index)(int, void ∗), pmath_string_t(∗underoverscriptbox_at_index)(int, void ∗), void(∗error)(pmath_string_t, int, void ∗, pmath_bool_t), void ∗data)

  Parses pMath code to a span array.

- pmath_t pmath_boxes_from_spans (pmath_span_array_t ∗spans, pmath_string_t string, pmath_bool_t parseable, pmath_t(∗box_at_index)(int, void ∗), void ∗data)

  Convert a span-array with the according code to boxed form.

- pmath_span_array_t ∗ pmath_spans_from_boxes (pmath_t boxes, pmath_string_t ∗result_string, void(∗make_box)(int, pmath_t, void ∗), void ∗data)

  Convert boxed form back to span-array and code.

- pmath_token_t pmath_token_analyse (const uint16_t ∗str, int len, int ∗prec)

Analyse a token.

- int pmath_token_prefix_precedence (const uint16_t ∗str, int len, int defprec)

  Give the prefix oprator precedence for a token.

- static PMATH_INLINE pmath_bool_t pmath_token_maybe_first (pmath_token_t tok)

  Test whether a token may be the first token in a subexpression.

- static PMATH_INLINE pmath_bool_t pmath_token_maybe_rest (pmath_token_t tok)

  Test whether a token need not be the first token in a subexpression.

- static PMATH_INLINE pmath_bool_t pmath_char_is_left (uint16_t ch)

  Test if a unicode character is a left bracket.

- static PMATH_INLINE uint16_t pmath_right_fence (uint16_t left)

  Get the corresponding right bracket to a given left bracket or 0.

- static PMATH_INLINE pmath_bool_t pmath_char_is_right (uint16_t ch)

  Test if a unicode character is a right bracket.

- static PMATH_INLINE pmath_bool_t pmath_char_is_name (uint16_t ch)

  Test if a unicode character can be the start of an identifier/name.

- static PMATH_INLINE pmath_bool_t pmath_char_is_integral (uint16_t ch)

  Test if a unicode character is an integral.

- static PMATH_INLINE pmath_bool_t pmath_token_maybe_bigop (pmath_token_t tok)

  Test if a token may be a big operator.

- static PMATH_INLINE pmath_bool_t pmath_char_maybe_bigop (uint16_t ch)

  Test if a unicode character may be a big operation, e.g. Union, Sum.

- static PMATH_INLINE pmath_bool_t pmath_char_is_digit (uint16_t ch)

  Test if a unicode character is a digit '0' - '9'.

- static PMATH_INLINE pmath_bool_t pmath_char_is_36digit (uint16_t ch)

  Test if a unicode character is a base-36 digit '0' - '9', 'a' - 'z', 'A' - 'Z'.

- static PMATH_INLINE pmath_bool_t pmath_char_is_basedigit (int base, uint16_t ch)

  Test if in a given base, a unicode character is a digit.

- static PMATH_INLINE pmath_bool_t pmath_char_is_hexdigit (uint16_t ch)

Test if a unicode character is a hexadecimal digit.

- void pmath_span_array_free (pmath_span_array_t ∗spans)

  Destroy a span-array and all its spans.

- int pmath_span_array_length (pmath_span_array_t ∗spans)

  Get a span-array's length.

- pmath_bool_t pmath_span_array_is_token_end (pmath_span_array_t ∗spans, int pos)

  Test the token-end-flag at an index.

- pmath_bool_t pmath_span_array_is_operand_start (pmath_span_array_t ∗spans, int pos)

  Test the operator-start-flag at an index.

- pmath_span_t ∗ pmath_span_at (pmath_span_array_t ∗spans, int pos)

  Get a span starting at an index.

- pmath_span_t ∗ pmath_span_next (pmath_span_t ∗span)

  Get the next-shorter span starting at the same position.

- int pmath_span_end (pmath_span_t ∗span)

  Get end of a span.

- pmath_t pmath_string_expand_boxes (pmath_string_t s)

  Expand a string that contains boxes to a list of Strings and Boxes.

- pmath_t pmath_parse_string (pmath_string_t code)

  Parse a string to an expression.

- pmath_t pmath_parse_string_args (const char ∗code, const char ∗format,...)

  Parse a string with additional arguments to an expression.

9.8.1 Detailed Description

Translating pMath code or boxes to pMath objects.

The pMath language supports standard mathematical notation (such as sums, ...). Therefor, code can be given as Boxes.

9.8.2 Example

The boxed form of $\sum_{i=1}^{n} f(i)$ is

```
{UnderoverscriptBox("\[Sum]", {"i", "=", "1"}, "n"),
{"f", "(", "i", ")"}}
```

or

```
{{"\[Sum]", SubsuperscriptBox({"i", "=", "1"}, "n")},
{"f", "(", "i", ")"}}
```

(\[Sum] is the Unicode character U+2211: "N-ARY SUMMATION").

It will be translated to `HoldComplete(Sum(f(i), i->1..n))` by `System`MakeExpression`.

Front-ends have to convert their own representation of the code to the boxed form before parsing.

9.8.3 Define Documentation

9.8.3.1 #define PMATH_CHAR_ALIASDELIMITER 0xF764

The character inserted by Richmath with ESCAPE or CAPSLOCK.

9.8.3.2 #define PMATH_CHAR_ALIASINDICATOR 0xF768

A character that looks like PMATH_CHAR_ALIASDELIMITER but has no effect.

9.8.3.3 #define PMATH_CHAR_ASSIGN 0x2254

The ":=" operator.

9.8.3.4 #define PMATH_CHAR_ASSIGNDELAYED 0x2A74

The "::=" operator.

9.8.3.5 #define PMATH_CHAR_BOX 0xFDD0

Represents a box.

9.8.3.6 #define PMATH_CHAR_INTEGRAL_D 0x2146

The integral "d".

9.8.3.7 #define PMATH_CHAR_INVISIBLECALL 0x2061

The Function application character.

9.8.3.8 #define PMATH_CHAR_LEFT_BOX 0xFFF9

Start of box code inside a string.

9.8.3.9 #define PMATH_CHAR_PIECEWISE 0xF361

The left curly bracket for cases.

9.8.3.10 #define PMATH_CHAR_PLACEHOLDER 0xFFFD

The Placeholder character. In richmath, type CAPSLOCK pl CAPSLOCK to insert it.

9.8.3.11 #define PMATH_CHAR_RIGHT_BOX 0xFFFB

End of box code inside a string.

9.8.3.12 #define PMATH_CHAR_RULE 0x2192

The " $\rightarrow$ " operator.

9.8.3.13 #define PMATH_CHAR_RULEDELAYED 0x29F4

The ":>" operator.

9.8.3.14 #define PMATH_CHAR_VECTOR 0x21C0

The arrow above names to indicate a vector.

9.8.3.15 #define PMATH_RUN(code)

Value:

```
pmath_unref( \
                pmath_evaluate( \
                                 pmath_parse_string( \
                                     PMATH_C_STRING(code))))
```

Execute some pMath code.

Parameters:

code The code as a C string (zero terminated).

9.8.3.16 #define PMATH_RUN_ARGS(code, format, ...)

Value:

```
pmath_unref( \
                pmath_evaluate( \
                                 pmath_parse_string_args( \
                                     (code), (format), __VA_ARGS__)))
```

Execute some pMath code with arguments.

Parameters:

code The code as a C string.

format The argument's type format string.

... The arguments.

See pmath_build_value() for the meaning of *format* and ...

9.8.4 Typedef Documentation

9.8.4.1 typedef struct _pmath_span_array_t pmath_span_array_t

9.8.4.2 typedef struct _pmath_span_t pmath_span_t

9.8.5 Enumeration Type Documentation

9.8.5.1 anonymous enum

Enumerator:

PMATH_PREC_ANY

PMATH_PREC_SEQ

PMATH_PREC_EVAL

PMATH_PREC_ASS

PMATH_PREC_MODY

PMATH_PREC_LAZY

PMATH_PREC_FUNC

PMATH_PREC_REPL

PMATH_PREC_RULE

PMATH_PREC_MAP

PMATH_PREC_STR

PMATH_PREC_COND

PMATH_PREC_ALT

PMATH_PREC_OR

PMATH_PREC_XOR

PMATH_PREC_AND

PMATH_PREC_ARROW

PMATH_PREC_REL

PMATH_PREC_UNION

PMATH_PREC_ISECT

PMATH_PREC_RANGE

PMATH_PREC_ADD

PMATH_PREC_CIRCADD

PMATH_PREC_PLUMI

PMATH_PREC_CIRCMUL

PMATH_PREC_MUL

PMATH_PREC_DIV

PMATH_PREC_MIDDOT

PMATH_PREC_CROSS

PMATH_PREC_MUL2

PMATH_PREC_POW

PMATH_PREC_FAC

PMATH_PREC_APL

PMATH_PREC_REPEAT

PMATH_PREC_TEST

PMATH_PREC_INC

PMATH_PREC_CALL

PMATH_PREC_DIFF

PMATH_PREC_PRIM

9.8.5.2 enum pmath_token_t

Token classes known in the pMath language.

Enumerator:

PMATH_TOK_NONE

PMATH_TOK_SPACE

PMATH_TOK_DIGIT

PMATH_TOK_STRING

PMATH_TOK_NAME

PMATH_TOK_NAME2

PMATH_TOK_BINARY_LEFT

PMATH_TOK_BINARY_RIGHT

PMATH_TOK_BINARY_LEFT_AUTOARG

PMATH_TOK_BINARY_LEFT_OR_PREFIX

PMATH_TOK_NARY

PMATH_TOK_NARY_AUTOARG

PMATH_TOK_NARY_OR_PREFIX

PMATH_TOK_POSTFIX_OR_PREFIX

PMATH_TOK_PREFIX

PMATH_TOK_POSTFIX

PMATH_TOK_CALL

PMATH_TOK_LEFTCALL

PMATH_TOK_LEFT

PMATH_TOK_RIGHT

PMATH_TOK_PRETEXT

PMATH_TOK_ASSIGNTAG

PMATH_TOK_PLUSPLUS

PMATH_TOK_COLON

PMATH_TOK_TILDES

PMATH_TOK_SLOT

PMATH_TOK_QUESTION

PMATH_TOK_INTEGRAL

PMATH_TOK_COMMENTEND

9.8.6 Function Documentation

9.8.6.1 pmath_t pmath_boxes_from_spans (pmath_span_array_t ∗ *spans*, pmath_string_t *string*, pmath_bool_t *parseable*, pmath_t(∗)(int, void ∗) *box_at_index*, void ∗ *data*)

Convert a span-array with the according code to boxed form.

Parameters:

spans A span-array. It can be obtained by pmath_spans_from_string() or pmath_spans_from_boxes().

string The corresponding code to *span*. It wont be freed.

parseable Whether whitespace and comments should be skipped or not.

box_at_index An optional function that returns the box at a given position (indicated by the PMATH_CHAR_BOX character). This function will be called (at most) one time for every box and in their order of apperance.

data A pointer that will be provided as the last argument to *box_at_index*.

Returns:

A pMath object representing the boxed form. It must be freed.

9.8.6.2 static PMATH_INLINE pmath_bool_t pmath_char_is_36digit (uint16_t *ch*) `[static]`

Test if a unicode character is a base-36 digit '0' - '9', 'a' - 'z', 'A' - 'Z'.

9.8.6.3 static PMATH_INLINE pmath_bool_t pmath_char_is_basedigit (int *base*, uint16_t *ch*) `[static]`

Test if in a given base, a unicode character is a digit.

9.8.6.4 static PMATH_INLINE pmath_bool_t pmath_char_is_digit (uint16_t *ch*) `[static]`

Test if a unicode character is a digit '0' - '9'.

9.8.6.5 static PMATH_INLINE pmath_bool_t pmath_char_is_hexdigit (uint16_t *ch*) `[static]`

Test if a unicode character is a hexadecimal digit.

9.8.6.6 static PMATH_INLINE pmath_bool_t pmath_char_is_integral (uint16_t *ch*) `[static]`

Test if a unicode character is an integral.

9.8.6.7 static PMATH_INLINE pmath_bool_t pmath_char_is_left (uint16_t *ch*) `[static]`

Test if a unicode character is a left bracket.

9.8.6.8 static PMATH_INLINE pmath_bool_t pmath_char_is_name (uint16_t *ch*) `[static]`

Test if a unicode character can be the start of an identifier/name.

9.8.6.9 static PMATH_INLINE pmath_bool_t pmath_char_is_right (uint16_t *ch*) `[static]`

Test if a unicode character is a right bracket.

9.8.6.10 static PMATH_INLINE pmath_bool_t pmath_char_maybe_bigop (uint16_t *ch*) `[static]`

Test if a unicode character may be a big operation, e.g. Union, Sum.

9.8.6.11 pmath_t pmath_parse_string (pmath_string_t *code*) `[related, inherited]`

Parse a string to an expression.

Parameters:

code A pMath String representing the code. It will be freed.

Returns:

A pMath object.

This function returns ToExpression("code"), but does not evaluate this released result.

9.8.6.12 pmath_t pmath_parse_string_args (const char ∗ *code*, const char ∗ *format*, ...) `[related, inherited]`

Parse a string with additional arguments to an expression.

Parameters:

code A pMath String representing the code. It will be freed.

format A format string for the arguments.

... The additional arguments.

Returns:

A pMath object.

This function is a short hand for

1. assigning pmath_build_value(format, ...) to the symbol $ParserArguments
2. then calling pmath_parse_string(PMATH_C_STRING(code)) and
3. restoring the old value of $ParserArguments
4. returning the result of the pmath_parse_string-call.

See also:

pmath_build_value
pmath_parse_string

9.8.6.13 static PMATH_INLINE uint16_t pmath_right_fence (uint16_t *left*) [static]

Get the corresponding right bracket to a given left bracket or 0.

9.8.6.14 void pmath_span_array_free (pmath_span_array_t ∗ *spans*) [inherited]

Destroy a span-array and all its spans.

Parameters:

spans The span-array.

9.8.6.15 pmath_bool_t pmath_span_array_is_operand_start (pmath_span_array_t ∗ *spans*, int *pos*) [inherited]

Test the operator-start-flag at an index.

Parameters:

spans The span-array.

pos The position. Must be between `0` and `pmath_span_array_-length(spans)-1`.

Returns:

Whether an operator starts at the specified position.

9.8.6.16 pmath_bool_t pmath_span_array_is_token_end (pmath_span_array_t ∗ *spans*, int *pos*) [inherited]

Test the token-end-flag at an index.

Parameters:

spans The span-array.

pos The position. Must be between `0` and `pmath_span_array_-length(spans)-1`.

Returns:

Whether a token ends at the specified position.

9.8.6.17 int pmath_span_array_length (pmath_span_array_t ∗ *spans*) `[inherited]`

Get a span-array's length.

Parameters:

spans The span-array.

Returns:

Its length or 0 if it's PMATH_NULL.

9.8.6.18 pmath_span_t ∗ pmath_span_at (pmath_span_array_t ∗ *spans*, int *pos*) `[inherited]`

Get a span starting at an index.

Parameters:

spans The span-array.

pos The position. Must be between `0` and `pmath_span_array_length(spans)-1`.

Returns:

The largest span stating at *pos* or PMATH_NULL if there is no span.

9.8.6.19 int pmath_span_end (pmath_span_t ∗ *span*) `[inherited]`

Get end of a span.

Parameters:

span A span.

Returns:

The last index which is covered by the span.

9.8.6.20 pmath_span_t ∗ pmath_span_next (pmath_span_t ∗ *span*) [inherited]

Get the next-shorter span starting at the same position.

Parameters:

span A span.

Returns:

The next-shorter span stating at *pos* or PMATH_NULL if there is no span.

9.8.6.21 pmath_span_array_t∗ pmath_spans_from_boxes (pmath_t *boxes*, pmath_string_t ∗ *result_string*, void(∗)(int, pmath_t, void ∗) *make_box*, void ∗ *data*)

Convert boxed form back to span-array and code.

Parameters:

boxes A pMath object representing the boxed form.

result_string A pointer where the resulting code will go to. Its previous value is ignored.

make_box A function that converts a boxed form (pMath object) to an actual box. It must free this object (the second argument).

data A pointer that will be provided to make_box as the last argument.

Returns:

A span-array. It must be freed with pmath_span_array_free() when it is no longer needed.

9.8.6.22 pmath_span_array_t∗ pmath_spans_from_string (pmath_string_t ∗ *code*, pmath_string_t(∗)(void ∗) *line_reader*, pmath_bool_t(∗)(int, void ∗) *subsuperscriptbox_at_index*, pmath_string_t(∗)(int, void ∗) *underoverscriptbox_at_index*, void(∗)(pmath_string_t, int, void ∗, pmath_bool_t) *error*, void ∗ *data*)

Parses pMath code to a span array.

Parameters:

code A pointer to a pMath string.

line_reader An optional function to be called, when there is more input needed. Its result will be appended to ∗code.

subsuperscriptbox_at_index An optional function that returns TRUE iff at a given position in the code (indicated by the PMATH_CHAR_BOX character) is a SubscriptBox, SuperscriptBox or SubsuperscriptBox.

underoverscriptbox_at_index [optional] If there is an UnderscriptBox, OverscriptBox or UnderoverscriptBox at a given position in the code (indicated by the PMATH_CHAR_BOX character) its base (e.g. middle part of UnderoverscriptBox) should be returned by this function, otherwise PMATH_NULL should be returned.

error A function that will be called on syntax errors. The first argument is `∗code`. It must not be freed. The second argument is the position in the code. The third argument is *data*. The fourth argument is TRUE if the error is critical and FALSE if it is just a warning (Syntax::newl) This function is optional, if it is PMATH_NULL, no messages will be generated during the scanning.

data An arbitrary pointer, that will be provided as the last argument to the callback functions.

Returns:

A span-array that can be used by pmath_boxes_from_spans to convert the code to boxed form, which, in turn, is used by System‘MakeExpression(). The span-array must be freed with pmath_span_array_free() when it is no longer needed.

9.8.6.23 pmath_t pmath_string_expand_boxes (pmath_string_t *s*) `[related, inherited]`

Expand a string that contains boxes to a list of Strings and Boxes.

Parameters:

s The string to be expanded. It will be freed.

Returns:

A string if there is nothing to expand or an expression representing s as boxes.

9.8.6.24 pmath_token_t pmath_token_analyse (const uint16_t ∗ *str*, int *len*, int ∗ *prec*)

Analyse a token.

Parameters:

str A UTF16-string.

len The length (in uint16_t-s) of the token *str*.

prec Optional address, where to store the default operator precedence for the token.

Returns:

The associated token class.

9.8.6.25 static PMATH_INLINE pmath_bool_t pmath_token_maybe_bigop (pmath_token_t *tok*) `[static]`

Test if a token may be a big operator.

9.8.6.26 static PMATH_INLINE pmath_bool_t pmath_token_maybe_first (pmath_token_t *tok*) `[static]`

Test whether a token may be the first token in a subexpression.

Parameters:

tok A token class.

Returns:

Whether tok may start a new subexpression

9.8.6.27 static PMATH_INLINE pmath_bool_t pmath_token_maybe_rest (pmath_token_t *tok*) `[static]`

Test whether a token need not be the first token in a subexpression.

Parameters:

tok A token class.

Returns:

Whether tok may reside inside a subexpression.

9.8.6.28 int pmath_token_prefix_precedence (const uint16_t ∗ *str*, int *len*, int *defprec*)

Give the prefix oprator precedence for a token.

Parameters:

str A UTF16-string.

len The length (in uint16_t-s) of the token *str*.

defprec The default operator precedence as given by pmath_token_analyse()

Returns:

The prefix operator precedence.

9.9 General Purpose Types

Useful type definitions that do not fit into any other category.

Defines

- #define FALSE ((pmath_bool_t)0)

  The FALSE value for pmath_bool_t.

- #define TRUE (!FALSE)

  The TRUE value for pmath_bool_t.

- #define PMATH_INVALID_PTR ((void∗)UINTPTR_MAX)

Typedefs

- typedef char pmath_bool_t

  A boolean type.

- typedef void(∗ pmath_callback_t)(void ∗)

  A general callback function.

9.9.1 Detailed Description

Useful type definitions that do not fit into any other category.

9.9.2 Define Documentation

9.9.2.1 #define FALSE ((pmath_bool_t)0)

The FALSE value for pmath_bool_t.

9.9.2.2 #define PMATH_INVALID_PTR ((void*)UINTPTR_MAX)

9.9.2.3 #define TRUE (!FALSE)

The TRUE value for pmath_bool_t.

9.9.3 Typedef Documentation

9.9.3.1 typedef char pmath_bool_t

A boolean type.

The C99 boolean type _Bool is not supported by all compilers, so we define a boolean type here for code clarity. The constants TRUE and FALSE can be used as return values for pmath_bool_t, but do not test on these. E.g. use `if(test)...` instead of `if(test == TRUE)...` but `return TRUE;` instead of `return 1;`

9.9.3.2 typedef void(* pmath_callback_t)(void *)

A general callback function.

This is used in various places where a callback function is needed, that does not only work with pMath objects.

9.10 Atomic Operations

Using atomic operations (independent of the rest of the library).

Defines

- #define PMATH_ATOMIC_FASTLOOP_COUNT (0)

  Loop iterations in spinlocks before yielding control.

- #define PMATH_DECLARE_ALIGNED(TYPE, NAME, ALIGNMENT) TYPE NAME

    Declares a variable with specified alignment.

Functions

- intptr_t pmath_atomic_fetch_add (pmath_atomic_t ∗atom, intptr_t delta)

    Add a value to another.

- intptr_t pmath_atomic_fetch_set (pmath_atomic_t ∗atom, intptr_t new_value)

    Exchange a value.

- intptr_t pmath_atomic_fetch_compare_and_set (pmath_atomic_t ∗atom, intptr_t old_value, intptr_t new_value)

    Exchange a value if it equals another value.

- pmath_bool_t pmath_atomic_compare_and_set (pmath_atomic_t ∗atom, intptr_t old_value, intptr_t new_value)

    Exchange a value if it equals another value.

- pmath_bool_t pmath_atomic_compare_and_set_2 (pmath_atomic2_t ∗atom, intptr_t old_value_fst, intptr_t old_value_snd, intptr_t new_value_fst, intptr_t new_value_snd)

    Exchange two values value if they equal another two values.

- pmath_bool_t pmath_atomic_have_cas2 (void)

    Check, whether the CPU supports pmath_atomic_compare_and_set_2().

- void pmath_atomic_barrier (void)

    Insert an explicit memory barrier.

- void pmath_atomic_lock (pmath_atomic_t ∗atom)

    Try to aquire a lock.

- void pmath_atomic_unlock (pmath_atomic_t ∗atom)

    Release a previously aquired lock.

- void pmath_atomic_loop_yield (void)

    Yield control to another thread (used in spinlocks).

- void pmath_atomic_loop_nop (void)

    A no-operation or short wait for use in spin locks.

9.10.1 Detailed Description

Using atomic operations (independent of the rest of the library).

pMath provides a collection of functions/macros to do atomic operations. This part of the library is completely independent of the rest of pMath. To use atomic opertions, #include <pmath-util/concurrency/atomic.h>. You do not have to link to an additionl library.

At the moment, supported compilers are GCC and Microsoft Visual C++.

On some platforms, the atomic opertations are implemented as inline functions with inline assembler (currently GCC older than 4.x). On other platforms, macros and compiler intrinsic functions (GCC 4.x, MSVC) are used.

9.10.2 Nice to read:

- http://developers.sun.com/solaris/articles/atomic_sparc/
- http://lists.canonical.org/pipermail/kragen-tol/1999-August/000457.html
- http://www.angstrom-distribution.org/unstable/sources/libc_sources.redhat.com__20061019.tar.gz

  libc/sysdeps/i386/i486/bits/atomic.h
- Intel's cmpxchg8b and cmpxchg16b instructions
- http://www.cse.msu.edu/~sdf/private/szumoframe-0.3.tar.gz

  szumoframe-0.3/src/szumoframe/szumo_preamble.h
- http://www.tml.tkk.fi/~rakajast/uvsr_renderer.tar.gz

  uvsr_renderer/needed_externals/threadlib/src/fifo.c
- qprof -> atomic_ops library
- http://code.google.com/p/google-perftools/

9.10.3 Define Documentation

9.10.3.1 #define PMATH_ATOMIC_FASTLOOP_COUNT (0)

Loop iterations in spinlocks before yielding control.

If the thread holding the lock sits on another CPU, spinning around a bit before pmath_atomic_loop_yield() reduces idle time. But if the thread holding the lock lives on the same CPU as the current thread (and thus is interrupted by the current thread), spinning elongates the wait time.

In summary, this should not be a compile time constant as it is now!

9.10.3.2 #define PMATH_DECLARE_ALIGNED(TYPE, NAME, ALIGNMENT) TYPE NAME

Declares a variable with specified alignment.

Parameters:

TYPE The variable type, possibly including the `volatile` modifier.

NAME The variable name.

ALIGNMENT The alignment in bytes

9.10.4 Function Documentation

9.10.4.1 void pmath_atomic_barrier (void)

Insert an explicit memory barrier.

9.10.4.2 pmath_bool_t pmath_atomic_compare_and_set (pmath_atomic_t * *atom*, intptr_t *old_value*, intptr_t *new_value*)

Exchange a value if it equals another value.

Parameters:

atom An atomic variable.

old_value The comparisor.

new_value The possible new value of `*atom`.

Returns:

Whether the exchange was performed.

This function compares `*atom` with `old_value` and iff both equal sets `*atom` to `new_value`, everything atomically and with aquire barrier semantics.

9.10.4.3 pmath_bool_t pmath_atomic_compare_and_set_2 (pmath_atomic2_t ∗ *atom*, intptr_t *old_value_fst*, intptr_t *old_value_snd*, intptr_t *new_value_fst*, intptr_t *new_value_snd*)

Exchange two values value if they equal another two values.

Parameters:

∗***atom*** An atomic variable of size 2 ∗ sizeof(void∗).

old_value_fst The first old value.

old_value_snd The second old value.

new_value_fst The possible new value of `atom`[0].

new_value_snd The possible new value of `atom`[1].

Returns:

Whether the exchange was performed or not.

This function compares `old_value_fst` with `atom`[0] and `old_value_snd` with atom[1]. If they equal, `atom`[0] is set to `new_value_fst` and `atom`[1] is set to `new_value_snd` and TRUE is returned. Otherwise, FALSE will be returned.

This function has aquire barrier semantics.

Note:

This function is not available on all Platforms. You must not call it if pmath_atomic_have_cas2() returns FALSE.

9.10.4.4 intptr_t pmath_atomic_fetch_add (pmath_atomic_t ∗ *atom*, intptr_t *delta*)

Add a value to another.

Parameters:

atom An atomic variable.

delta The difference between the new and the old value.

Returns:

The old value of ∗`atom`.

This function increments ∗`atom` atomically by `delta`. It has full memory barrier semantics.

9.10.4.5 intptr_t pmath_atomic_fetch_compare_and_set (pmath_atomic_t * *atom*, intptr_t *old_value*, intptr_t *new_value*)

Exchange a value if it equals another value.

Parameters:

atom An atomic variable.

old_value The comparisor.

new_value The possible new value of `*atom`.

Returns:

The old value of `*atom`.

You should use pmath_atomic_compare_and_set() if you don't need the exact old value of `*atom`, because this function might be non-existent on some systems. This function has aquire barrier semantics.

9.10.4.6 intptr_t pmath_atomic_fetch_set (pmath_atomic_t * *atom*, intptr_t *new_value*)

Exchange a value.

Parameters:

atom An atomic variable.

new_value The new value of `*atom`.

Returns:

The old value of `*atom`.

This function sets `*atom` to `new_value` and returns the old value atomically. It has full memory barrier semantics.

9.10.4.7 pmath_bool_t pmath_atomic_have_cas2 (void)

Check, whether the CPU supports pmath_atomic_compare_and_set_2().

Returns:

whether pmath_atomic_compare_and_set_2() is supported.

Note, that a call to pmath_atomic_compare_and_set_2() will crash your application on any platform that does not support the operation (e.g. pre-Pentiums, early AMD64).

9.10.4.8 void pmath_atomic_lock (pmath_atomic_t * *atom*)

Try to aquire a lock.

Parameters:

atom The lock. An atomic variable.

This function implements a spin lock. It has aquire barrier semantics. Use it with pmath_atomic_unlock():

```
pmath_atomic_t spin = PMATH_ATOMIC_STATIC_INIT;
...
pmath_atomic_lock(&spin)
... critical section ...
pmath_atomic_unlock(&spin);
```

9.10.4.9 void pmath_atomic_loop_nop (void)

A no-operation or short wait for use in spin locks.

9.10.4.10 void pmath_atomic_loop_yield (void)

Yield control to another thread (used in spinlocks).

9.10.4.11 void pmath_atomic_unlock (pmath_atomic_t * *atom*)

Release a previously aquired lock.

Parameters:

atom The lock. An atomic variable.

See also:

pmath_atomic_lock

9.11 Thread Messaging

Sending messages to other threads.

Data Structures

- class pmath_messages_t

  A message queue for interthread communication.

Typedefs

- typedef pmath_custom_t pmath_messages_t

Functions

- double pmath_tickcount (void)

  Gives the seconds since January 1, 1970 (UTC).

- pmath_bool_t pmath_is_message_queue (pmath_t obj)

  Test if an object is a message queue.

- pmath_messages_t pmath_thread_get_queue (void)

  Get the current thread's message queue.

- void pmath_thread_sleep (void)

  Send the current thread to sleep.

- void pmath_thread_sleep_timeout (double abs_timeout)

  Send the current thread to sleep.

- void pmath_thread_wakeup (pmath_messages_t mq)

  Wake up another thread.

- void pmath_thread_send (pmath_messages_t mq, pmath_t msg)

  Asynchronously send a message to another thread.

- pmath_t pmath_thread_send_wait (pmath_messages_t mq, pmath_t msg, double timeout_seconds, void(∗idle_function)(void ∗), void ∗idle_data)

  Send a message to another thread and wait for the answer.

- void pmath_thread_send_delayed (pmath_messages_t mq, pmath_t msg, double seconds)

  Asynchronously send a message to a thread sometime in the future.

9.11.1 Detailed Description

Sending messages to other threads.

Every pMath thread has its own message queue. Other threads can send messages to such a queue and optionally wait for a result. Messages to any queue can also be registered for delivery at a later point in time.

Threads can go to sleep when they have no work to do. They will be awaken any time a message arrives to handle it.

Technical Note: Pending messages are handled as soon as time pmath_aborting() is called, which happens periodically. For the pMath code, it looks like asynchronous signals, because messages can occur any time during the evaluation. But from the native code's point of view, messages are synchronous, because they can only occur during pmath_aborting().

Note:

> Message passing is not signal-safe. You must not send any messages from within a UNIX signal handler.

9.11.2 Typedef Documentation

9.11.2.1 typedef pmath_custom_t pmath_messages_t

9.11.3 Function Documentation

9.11.3.1 pmath_bool_t pmath_is_message_queue (pmath_t *obj*) `[related, inherited]`

Test if an object is a message queue.

Parameters:

> ***obj*** Any pMath object. It wont be freed.

Returns:

> TRUE if the object is a valid message queue object (pmath_messages_t).

9.11.3.2 pmath_messages_t pmath_thread_get_queue (void) `[related, inherited]`

Get the current thread's message queue.

Returns:

A refernce to the message queue or PMATH_NULL on error. You must destroy it with pmath_unref() when its no longer needed.

9.11.3.3 void pmath_thread_send (pmath_messages_t *mq*, pmath_t *msg*) `[related, inherited]`

Asynchronously send a message to another thread.

Parameters:

mq The receivers message queue. It wont be freed.

msg The message. It will be freed.

The message will be evaluated by the receiver. This function returns immediately. If the receiver cannot handle the message (since it is dead or there is not enough memory), the message will be deleted.

Note that messages might not be handled in the order they were send.

9.11.3.4 void pmath_thread_send_delayed (pmath_messages_t *mq*, pmath_t *msg*, double *seconds*) `[related, inherited]`

Asynchronously send a message to a thread sometime in the future.

Parameters:

mq The receivers message queue. It wont be freed.

msg The message. It will be freed.

seconds The delay in seconds before the message will be delivered.

The message will be evaluated by the receiver. This function returns immediately. If the receiver cannot handle the message (since it is dead or there is not enough memory), the message will be deleted.

9.11.3.5 pmath_t pmath_thread_send_wait (pmath_messages_t *mq*, pmath_t *msg*, double *timeout_seconds*, void(∗)(void ∗) *idle_function*, void ∗ *idle_data*) `[related, inherited]`

Send a message to another thread and wait for the answer.

Parameters:

mq The receivers message queue. It wont be freed.

msg The message. It will be freed.

timeout_seconds The maximum number of seconds tho wait for the answer. Use HUGE_VAL if you do not want a timeout.

idle_function An optional function that will be called any time the waiting thread wakes up but there is no answer yet.

idle_data Argument for

- idle_function.

Returns:

The result of pmath_evaluate(message) called by the receiver or PMATH_-UNDEFINED in case of an error.

The message will be evaluated by the receiver. If the receiver cannot handle it (since it is dead or there is not enough memory), the message will be deleted.

The calling thread will fall asleep until

- it receives an answer to return or
- the message is deleted or
- the timeout is reached or
- another abort situation occurs in the calling thread (e.g. pmath_abort_please() is called anywhere in the system)

In the last two cases (timeout or abort), a the remote evaluation will be aborted.

Todo

Check, what happens if mq belongs to a parent thread.

9.11.3.6 void pmath_thread_sleep (void) [related, inherited]

Send the current thread to sleep.

The thread will fall asleep until

- it receives a message or
- it is waken up with pmath_thread_wakeup() or
- an abort-condition (pmath_abort_please() or pmath_throw()) is met *anywhere* in the system.

Because of the last point, this function is normally called in a loop:

```
while(!pmath_aborting() && some_wait_condition){
  pmath_thread_sleep();
}
```

9.11.3.7 void pmath_thread_sleep_timeout (double *abs_timeout*) `[related, inherited]`

Send the current thread to sleep.

Parameters:

abs_timeout Timeout in seconds since January 1, 1970 (UTC).

The thread will fall asleep until

- it receives a message or
- it is waken up with pmath_thread_wakeup() or
- an abort-condition (pmath_abort_please() or pmath_throw()) is met *anywhere* in the system or
- - abs_timeout is passed.

  See also:

  pmath_thread_sleep, pmath_tickcount

9.11.3.8 void pmath_thread_wakeup (pmath_messages_t *mq*) `[related, inherited]`

Wake up another thread.

Parameters:

mq The message queue associated with the sleeping thread. It wont be freed.

This function wakes up the thread that is associated with the message queue. It is safe to try to wake up threads, that are not sleeping.

To follow the loop-style waiting idiom described in pmath_thread_sleep(), you must modify `some_wait_condition` *before* calling this function to successfully awake the other thread.

9.11.3.9 double pmath_tickcount (void)

Gives the seconds since January 1, 1970 (UTC).

Returns:

The number of seconds since January 1, 1970 (UTC)

9.12 Multithreading with pMath

The Thread abstraction in pMath.

Data Structures

- class pmath_threadlock_t

  A reentrant lock for threads.

- class pmath_thread_t

  The Representation of a thread.

Typedefs

- typedef struct _pmath_threadlock_t ∗ pmath_threadlock_t
- typedef struct _pmath_thread_t ∗ pmath_thread_t

Functions

- pmath_t pmath_thread_local_save (pmath_t key, pmath_t value)

  Store a thread/thread-local value.

- pmath_t pmath_thread_local_load (pmath_t key)

  Load a thread/thread-local value.

- void pmath_throw (pmath_t exception)

  Throw an exception.

- pmath_t pmath_catch (void)

  Catch any exception.

- pmath_bool_t pmath_aborting (void)

  Queries whether pMath was requested to abort the evaluation of the current thread.

- void pmath_abort_please (void)

  Requests pMath to abort the current evaluation.

- void pmath_suspend_all_please (void)

  Suspend all other threads. This function does not realy suspend threads immediately. Any other thread, that calls pmath_aborting() (or pmath_thread_aborting()), will block until we call pmath_resume_all().

- void pmath_resume_all (void)

  Resume all other threads.

- void pmath_thread_call_locked (pmath_threadlock_t *threadlock_ptr, pmath_callback_t callback, void *data)

  Execute a function synchronized with a threadlock.

- pmath_bool_t pmath_thread_queue_is_blocked_by (pmath_messages_t waiter_mq, pmath_messages_t waitee_mq)

  Queries whether a thread is blocked by another thread.

- void pmath_thread_run_with_interrupt_notifier (pmath_callback_t callback, pmath_callback_t notify, void *callback_closure, void *notify_closure)

  Execute a function with an interrupt notifier installed.

- pmath_thread_t pmath_thread_get_current (void)

  Get the current pMath thread.

- pmath_thread_t pmath_thread_get_parent (pmath_thread_t thread)

  Get a thread's direct parent.

- pmath_bool_t pmath_thread_is_parent (pmath_thread_t parent, pmath_thread_t child)

  Queries whether a thread is one of the parents of another.

- pmath_bool_t pmath_thread_aborting (pmath_thread_t thread)

  Queries whether pMath was requested to abort the evaluation of a specific thread or its parents.

9.12.1 Detailed Description

The Thread abstraction in pMath.

pMath stores several data local to a thread. Therefor, it maintains a pmath_thread_t in every operating system thread it runs on. Those pmath_thread_t variables are created and freed via pmath_init() and pmath_done() respectively. Thus, you have to call those two functions once in every thread that uses pMath functions (and abort the thread if pmath_init() fails).

pMath Threads can have parents. While one thread is running, its parent thread waits (for all its children) and is effectively immutable. This way, child threads can read their parent thread's local variables.

9.12.2 Synchronization

In other environments, you normaly do synchronization with mutexes and the like. But if we did so, a deadlock could occur when a mutex is allready locked by the parent thread, which in turn is waiting for its children to finish.

The solution is to use pMath threadlocks: You simply synchronize with a pmath_symbol_t through pmath_symbol_synchronized() or directly with a pmath_threadlock_t and pmath_thread_call_locked(). This is reentrant and locks execution to a given thread *and* its child threads. pMath cares about avoiding deadlocks behind the scenes.

Note that threadlocks are needed only if the syncronized code might create child threads or calls other code that utilizes thread locks. Threads might be created by pmath_evaluate() & co.

In other situations, you should use mutexes/semaphores from your operating system library or spinlocks (see pmath_atomic_lock() and pmath_atomic_unlock()), because they are much faster.

For some simple changes on global integer/pointer variables, you can use Atomic Operations.

9.12.3 Typedef Documentation

9.12.3.1 typedef struct _pmath_thread_t∗ pmath_thread_t

9.12.3.2 typedef struct _pmath_threadlock_t∗ pmath_threadlock_t

9.12.4 Function Documentation

9.12.4.1 void pmath_abort_please (void)

Requests pMath to abort the current evaluation.

This function is signal-safe.

See also:

pmath_continue_after_abort()

9.12.4.2 pmath_bool_t pmath_aborting (void)

Queries whether pMath was requested to abort the evaluation of the current thread.

Returns:

Whether the user called pmath_abort_please() or an exception was thrown or a time-out is passed.

9.12.4.3 pmath_t pmath_catch (void)

Catch any exception.

Returns:

exception The exception to be thrown. If there is no exception available, PMATH_-UNDEFINED will be returned.

If you cannot handle the exception, you can re-throw it with pmath_throw().

9.12.4.4 void pmath_resume_all (void)

Resume all other threads.

See also:

pmath_suspend_all_please

9.12.4.5 void pmath_suspend_all_please (void)

Suspend all other threads. This function does not realy suspend threads immediately. Any other thread, that calls pmath_aborting() (or pmath_thread_aborting()), will block until we call pmath_resume_all().

9.12.4.6 pmath_bool_t pmath_thread_aborting (pmath_thread_t *thread*) [related, inherited]

Queries whether pMath was requested to abort the evaluation of a specific thread or its parents.

Parameters:

thread A thread that should be tested.

Returns:

Whether the given thread should abort evaluation.

See also:

pmath_aborting

9.12.4.7 void pmath_thread_call_locked (pmath_threadlock_t ∗ *threadlock_ptr*, pmath_callback_t *callback*, void ∗ *data*) `[related, inherited]`

Execute a function synchronized with a threadlock.

Parameters:

threadlock_ptr A pointer to the threadlock.

callback The function to be executed when the symbol is locked.

data A pointer that will be passed to callback.

All you have to do is initialize the threadlock `threadlock_ptr` points to with NULL before you call this function for the first time:

```
static pmath_threadlock_t lock = NULL;
...
pmath_thread_call_locked(&lock, my_callback, my_data);
```

To synchronize with a symbol, use pmath_symbol_synchronized().

9.12.4.8 pmath_thread_t pmath_thread_get_current (void) `[related, inherited]`

Get the current pMath thread.

Returns:

A pmath_thread_t. This is PMATH_NULL, if you did not register the current thread to pMath via pmath_init().

9.12.4.9 pmath_thread_t pmath_thread_get_parent (pmath_thread_t *thread*) `[related, inherited]`

Get a thread's direct parent.

Parameters:

thread A thread.

Returns:

The direct parent of thread. Usualy PMATH_NULL.

9.12.4.10 pmath_bool_t pmath_thread_is_parent (pmath_thread_t *parent*, pmath_thread_t *child*) `[related, inherited]`

Queries whether a thread is one of the parents of another.

Parameters:

parent A thread.

child A thread.

Returns:

TRUE, if parent is a parent thread of child or if parent == child. FALSE otherwise.

It is important to know that a parent thread is never executed in parallel with its children. However, to check for threads that depend on *child* (e.q. to evaluate a funciton on a specific thread or any thread that it waits on), use pmath_thread_queue_is_blocked_by().

9.12.4.11 pmath_t pmath_thread_local_load (pmath_t *key*)

Load a thread/thread-local value.

Parameters:

key A key that was used to save the value with _pmath_thread_local_save() before. It wont be freed.

Returns:

PMATH_UNDEFINED or the stored value. You must destroy the it.

If there is nothing stored for key in the current thread, its parent threads are processed. If none of them stores something under 'key' and key is a symbol, The global value is used.

9.12.4.12 pmath_t pmath_thread_local_save (pmath_t *key*, pmath_t *value*)

Store a thread/thread-local value.

Parameters:

key The key that can be used to obtain the value with _pmath_thread_local_load(). It wont be freed.

value The thread/thread-local value. It will be freed.

Returns:

PMATH_UNDEFINED or the previous value that was stored with the same key. You must destroy it.

Note that keys of the form 'symboltag' are used to store whether a message should be suppressed (value PMATH_SYMBOL_OFF) or not (value PMATH_NULL).

All keys that are magic numbers, have special meanings for pmath_thread_local_save(). You should not use them as the a key.

Keys which are only symbols are used for thread-local symbols (

See also:

pmath_symbol_attributes_t).

9.12.4.13 pmath_bool_t pmath_thread_queue_is_blocked_by (pmath_messages_t *waiter_mq*, pmath_messages_t *waitee_mq*) `[related, inherited]`

Queries whether a thread is blocked by another thread.

Parameters:

waiter_mq A message queue. It will be freed

waitee_mq A message queue. It will be freed.

Returns:

TRUE, if thread which owns waiter_mq is a blocked by the thread which owns waitee_mq or if waiter_mq == waitee_mq. FALSE otherwise.

A use-case for this function is a function that wants to be evaluated on a specific thread A or any thread that A waits on. See builtin_interrupt() in the reference front-end implementation test.exe

9.12.4.14 void pmath_thread_run_with_interrupt_notifier (pmath_callback_t *callback*, pmath_callback_t *notify*, void ∗ *callback_closure*, void ∗ *notify_closure*) `[related, inherited]`

Execute a function with an interrupt notifier installed.

Parameters:

callback The function to be called.

notify Is called when a message is delivered to the current thread. This function is called by the thread that sends the message, but concurrent sending threads wait on each other when calling the function. *notify* must not call any function that could send messages, because that would lead to a deadlock.

callback_closure The argument for *callback*.

notify_closure The argument for *notify*.

9.12.4.15 void pmath_throw (pmath_t *exception*)

Throw an exception.

Parameters:

exception The exception to be thrown. It will be freed. You cannot throw the magic number PMATH_UNDEFINED.

If there is already an uncought exception, this new exception is lost.

9.13 Debugging

Functions

- void pmath_debug_print (const char ∗fmt,...)

  Print out a simple debug message.

- void pmath_debug_print_object (const char ∗pre, pmath_t obj, const char ∗post)

  Print a pMath object to the debug log.

- void pmath_debug_print_stack (void)

  Print the current pMath stack trace to the debug log.

9.13.1 Detailed Description

These functions are for logging purposes. They default to ((void)0) unless `PMATH_DEBUG_LOG` is defined.

9.13.2 Function Documentation

9.13.2.1 void pmath_debug_print (const char ∗ *fmt*, ...)

Print out a simple debug message.

Parameters:

fmt A printf-compatible format string

... The variables to be printed as specified by format.

The format string and arguments are as in printf.

9.13.2.2 void pmath_debug_print_object (const char ∗ *pre*, pmath_t *obj*, const char ∗ *post*)

Print a pMath object to the debug log.

Parameters:

pre A string that should be printed before the object.

obj A pMath object. It wont be freed.

post A string that should be printed after the object.

9.13.2.3 void pmath_debug_print_stack (void)

Print the current pMath stack trace to the debug log.

9.14 File API

Unified API to access file or other memory content.

Data Structures

- class pmath_binary_file_api_t

  Access functions for binary files.

- class pmath_text_file_api_t

  Access functions for text files.

Enumerations

- enum pmath_files_status_t {
  PMATH_FILE_OK = 0, PMATH_FILE_INVALID = 1,
  PMATH_FILE_ENDOFFILE = 2, PMATH_FILE_OTHERERROR = 3,
  PMATH_FILE_RECURSIVE = 4 }

  The status of a file.

Functions

- pmath_symbol_t pmath_file_create_compressor (pmath_t dstfile)

  Create a writeable binary file object that compresses its input.

- pmath_symbol_t pmath_file_create_uncompressor (pmath_t srcfile)

  Create a readable binary file object that uncompresses its input.

- pmath_bool_t pmath_file_test (pmath_t file, int properties)

  Check whether a file supports a set of properties.

- pmath_files_status_t pmath_file_status (pmath_t file)

  Get the current status of a readable file.

- size_t pmath_file_read (pmath_t file, void ∗buffer, size_t buffer_size, pmath_bool_t preserve_internal_buffer)

  Read some bytes from a binary file.

- pmath_string_t pmath_file_readline (pmath_t file)

  Read one line from a text file.

- void pmath_file_set_textbuffer (pmath_t file, pmath_string_t buffer)

  Set a file's internal text buffer.

- size_t pmath_file_write (pmath_t file, const void ∗buffer, size_t buffer_size)

  Write some bytes to a binary file.

- pmath_bool_t pmath_file_writetext (pmath_t file, const uint16_t ∗str, int len)

  Write to a text file.

- void pmath_file_flush (pmath_t file)

  Flush the output buffer of a writeable file.

- pmath_bool_t pmath_file_write_object (pmath_t file, pmath_t obj, pmath_write_options_t options)

  Write an object to a text file.

- pmath_bool_t pmath_file_set_binbuffer (pmath_t file, size_t size)

  Set a binary file's buffer size.

- void pmath_file_manipulate (pmath_t file, void(∗type)(void ∗), void(∗callback)(void ∗, void ∗), void ∗data)

  Manipulate a file's internal representation.

- pmath_bool_t pmath_file_close (pmath_t file)

  Closes a file.

- void pmath_file_close_if_unused (pmath_t file)

  Closes a file if it is not referenced somewhere else.

- pmath_symbol_t pmath_file_create_binary (void ∗extra, void(∗extra_destructor)(void ∗), pmath_binary_file_api_t ∗api)

  Create a binary file object.

- pmath_symbol_t pmath_file_create_text (void ∗extra, void(∗extra_destructor)(void ∗), pmath_text_file_api_t ∗api)

  Create a text file object.

- pmath_symbol_t pmath_file_create_text_from_binary (pmath_t binfile, const char ∗encoding)

  Create a text file object operating on a binary file.

- pmath_symbol_t pmath_file_create_binary_buffer (size_t mincapacity)

  Create a byte-stream file object.

- size_t pmath_file_binary_buffer_size (pmath_t binfile)

  Get The number of readable bytes in a binary buffer.

- void pmath_file_binary_buffer_manipulate (pmath_t binfile, void(∗callback)(uint8_t ∗readable, uint8_t ∗∗writable, const uint8_t ∗end, void ∗closure), void ∗closure)

  Manipulate the content of a binary buffer.

- void pmath_file_create_mixed_buffer (const char ∗encoding, pmath_symbol_t ∗out_textfile, pmath_symbol_t ∗out_binfile)

  Creates a mixed binary/text file double ended queue.

9.14.1 Detailed Description

Unified API to access file or other memory content.

A file is a Symbol (normally with attribute PMATH_SYMBOL_ATTRIBUTE_TEMPORARY) whose value is a special Custom Object.

The output functions can operate on lists of files.

9.14.2 Enumeration Type Documentation

9.14.2.1 enum pmath_files_status_t

The status of a file.

See also:

pmath_file_status()

Enumerator:

PMATH_FILE_OK No error.

PMATH_FILE_INVALID The object is no readable file.

PMATH_FILE_ENDOFFILE The (readable) file position is at the end.

PMATH_FILE_OTHERERROR There is another problem with the file.

PMATH_FILE_RECURSIVE The file is already locked by the current thread.

9.14.3 Function Documentation

9.14.3.1 void pmath_file_binary_buffer_manipulate (pmath_t *binfile*, void(∗)(uint8_t ∗readable, uint8_t ∗∗writable, const uint8_t ∗end, void ∗closure) *callback*, void ∗ *closure*)

Manipulate the content of a binary buffer.

Parameters:

binfile A binary file created with pmath_file_create_binary_buffer(). It wont be freed.

callback The callback function that does the manipulation.

closure The fourth parameter for *callback*.

The *callback* function must not write before *readable* or after *end*. ∗*writable* gives the current write-position, which is always between *readable* and *end*. It can be changed insidethe callback, but must remain between *readable* and *end*.

9.14.3.2 size_t pmath_file_binary_buffer_size (pmath_t *binfile*)

Get The number of readable bytes in a binary buffer.

Parameters:

binfile A binary file created with pmath_file_create_binary_buffer(). It wont be freed.

Returns:

The number of readable bytes in the binary buffer or 0 on error.

9.14.3.3 pmath_bool_t pmath_file_close (pmath_t *file*)

Closes a file.

Parameters:

file A file object. It will be freed.

Returns:

Whether file was a valid file object.

Files are closed automatically by the garbage collector when the reference counter becomes zero, which could be at an unpredictable point time in the future. This function closes a file immediatly (by clearing the value of the symbol representing the file).

9.14.3.4 void pmath_file_close_if_unused (pmath_t *file*)

Closes a file if it is not referenced somewhere else.

Parameters:

file A file object. It will be freed.

See also:

pmath_file_close

9.14.3.5 pmath_symbol_t pmath_file_create_binary (void $*$ *extra*, void($*$)(void $*$) *extra_destructor*, pmath_binary_file_api_t $*$ *api*)

Create a binary file object.

Parameters:

extra Arbitrary extra data.

extra_destructor A function to destroy the extra data.

api The file access functions.

Returns:

A newly created binary file object. You can destroy and close it with pmath_file_close() or pmath_unref().

The *api* functions are never called by more than one thread at once. This is assured with a non-reentrant spinlock.

See also:

pmath_binary_file_api_t

9.14.3.6 pmath_symbol_t pmath_file_create_binary_buffer (size_t *mincapacity*)

Create a byte-stream file object.

Parameters:

mincapacity The initial size of the buffer.

Returns:

A newly created binary file object. You can destroy and close it with pmath_file_close() or pmath_unref().

You can write to a byte-buffer and read previously written data from it. Note that this does not support random access to the data.

9.14.3.7 pmath_symbol_t pmath_file_create_compressor (pmath_t *dstfile*)

Create a writeable binary file object that compresses its input.

Parameters:

dstfile A writable binary file object to write the compressed data to. It will be freed.

Returns:

A writeable binary file object or PMATH_NULL on error.

The compression uses zlib.

9.14.3.8 void pmath_file_create_mixed_buffer (const char ∗ *encoding*, pmath_symbol_t ∗ *out_textfile*, pmath_symbol_t ∗ *out_binfile*)

Creates a mixed binary/text file double ended queue.

Parameters:

encoding The encoding name. Possible values are "latin1", and "base85"

out_textfile Will be set to the readable/writable text end.

out_binfile Will be set to the readable/writable binary end.

9.14.3.9 pmath_symbol_t pmath_file_create_text (void $*$ *extra*, void($*$)(void $*$) *extra_destructor*, pmath_text_file_api_t $*$ *api*)

Create a text file object.

Parameters:

extra Arbitrary extra data.

extra_destructor A function to destroy the extra data.

api The file access functions.

Returns:

A newly created text file object. You can destroy and close it with pmath_file_close() or pmath_unref().

The *api* functions are never called by more than one thread at once. This is assured with a non-reentrant spinlock.

See also:

pmath_text_file_api_t

9.14.3.10 pmath_symbol_t pmath_file_create_text_from_binary (pmath_t *binfile*, const char $*$ *encoding*)

Create a text file object operating on a binary file.

Parameters:

binfile A binary file. It will be freed.

encoding A text encoding that the iconv library knows.

Returns:

A newly created text file object. You can destroy and close it with pmath_file_close() or pmath_unref().

9.14.3.11 pmath_symbol_t pmath_file_create_uncompressor (pmath_t *srcfile*)

Create a readable binary file object that uncompresses its input.

Parameters:

srcfile A readable binary file object to read the compressed data from. It will be freed.

Returns:

A readable binary file object or PMATH_NULL on error.

The uncompression uses zlib.

9.14.3.12 void pmath_file_flush (pmath_t *file*)

Flush the output buffer of a writeable file.

Parameters:

file A writeable binary file object. It wont be freed.

9.14.3.13 void pmath_file_manipulate (pmath_t *file*, void(∗)(void ∗) *type*, void(∗)(void ∗, void ∗) *callback*, void ∗ *data*)

Manipulate a file's internal representation.

Parameters:

file A file object. It wont be freed.

type The *extra_destructor* that was provided to pmath_file_create_binary() or pmath_file_create_text().

callback A callback function. The first argument will be the *extra* parameter that was provided to pmath_file_create_binary() or pmath_file_create_text().

data The second parameter for *callback*.

If *file* is a valid file object and if *type* is the *extra_destructor* which *file* was created with, then and only then *callback* will be called.

This function does not support lists of writeable files.

9.14.3.14 size_t pmath_file_read (pmath_t *file*, void ∗ *buffer*, size_t *buffer_size*, pmath_bool_t *preserve_internal_buffer*)

Read some bytes from a binary file.

Parameters:

file A readable binary file object. It wont be freed.

buffer The read bytes will go here.

buffer_size The number of bytes you would like to read/size of *buffer*.

preserve_internal_buffer If TRUE, a subsequent call will get the same buffer content. If FALSE, the file pointer will be moved.

Returns:

Number of read bytes. This is never more than *buffer_size*. If *preserve_internal_-buffer* is TRUE or in case of an error, this can be less than *buffer_size*.

9.14.3.15 pmath_string_t pmath_file_readline (pmath_t *file*)

Read one line from a text file.

Parameters:

file A readable text file object. It wont be freed.

Returns:

The next line in the file.

9.14.3.16 pmath_bool_t pmath_file_set_binbuffer (pmath_t *file*, size_t *size*)

Set a binary file's buffer size.

Parameters:

file A binary file. It wont be freed.

size The new size of the buffer in bytes.

Returns:

TRUE if the operation succeded.

This function might clear the old buffer. So it should be called before any file read operation is done.

9.14.3.17 void pmath_file_set_textbuffer (pmath_t *file*, pmath_string_t *buffer*)

Set a file's internal text buffer.

Parameters:

file A readable text file. It wont be freed.

buffer A string. It should not contain any new line characters. It will be freed.

9.14.3.18 pmath_files_status_t pmath_file_status (pmath_t *file*)

Get the current status of a readable file.

Parameters:

file A readable file object. It wont be freed.

Returns:

pmath_files_status_t The current file status.

9.14.3.19 pmath_bool_t pmath_file_test (pmath_t *file*, int *properties*)

Check whether a file supports a set of properties.

Parameters:

file A file object. It wont be freed.

properties File properties. See remarks section.

Returns:

TRUE iff the file supports all of the requested properties.

Remarks:

properties can be zero or more of the following values:

- `PMATH_FILE_PROP_READ`: The file is readable.
- `PMATH_FILE_PROP_WRITE`: The file is writeable.
- `PMATH_FILE_PROP_BINARY`: It is a binary file.
- `PMATH_FILE_PROP_TEXT`: It is a text file.

Lists of writeable files are writeable, too. Only Symbols can be readable files.

9.14.3.20 size_t pmath_file_write (pmath_t *file*, const void ∗ *buffer*, size_t *buffer_size*)

Write some bytes to a binary file.

Parameters:

file A writeable binary file object. It wont be freed.

buffer The data to be written.

buffer_size The number of bytes to write/size of *buffer*.

Returns:

The number of written bytes. This is less than buffer_size in case of an error. If file is a list of files this is the smallest number of written bytes to the single files.

9.14.3.21 pmath_bool_t pmath_file_write_object (pmath_t *file*, pmath_t *obj*, pmath_write_options_t *options*)

Write an object to a text file.

Parameters:

file A writeable text file object. It wont be freed.

obj An object. It wont be freed.

options Some options defining the format.

Returns:

Whether the operation succeeded.

9.14.3.22 pmath_bool_t pmath_file_writetext (pmath_t *file*, const uint16_t ∗ *str*, int *len*)

Write to a text file.

Parameters:

file A writeable text file object. It wont be freed.

str A UTF-16 string buffer. e.g. pmath_string_buffer(&some_text)

len The number of uint16_t in the buffer. e.g. pmath_string_length(some_text). This can be -1 if *str* is zero terminated.

Returns:

Whether the operation succeeded.

9.15 Hashtables

A general hashtable implementation.

Data Structures

- class pmath_ht_class_t

  A hashtable interface.

- class pmath_hashtable_t

  The Hashtable class.

Typedefs

- typedef struct _pmath_hashtable_t ∗ pmath_hashtable_t
- typedef void(∗ pmath_ht_entry_callback_t)(void ∗entry, void ∗data)

  A callback function for hashtable entries.

- typedef void ∗(∗ pmath_ht_entry_copy_t)(void ∗entry)

  An entry copy function.

- typedef unsigned int(∗ pmath_ht_entry_hash_func_t)(void ∗entry)

  An entry hash function.

- typedef unsigned int(∗ pmath_ht_key_hash_func_t)(void ∗key)

  A key hash function.

- typedef pmath_bool_t(∗ pmath_ht_entry_equal_func_t)(void ∗entry1, void ∗entry2)

  An entry comparision function.

- typedef pmath_bool_t(∗ pmath_ht_entry_equals_key_func_t)(void ∗entry, void ∗key)

  An entry to key comparision function.

Functions

- pmath_hashtable_t pmath_ht_create (const pmath_ht_class_t ∗klass, unsigned int minsize)

  Create a new hashtable.

- pmath_hashtable_t pmath_ht_copy (pmath_hashtable_t ht, pmath_ht_entry_copy_t entry_copy)

  Copy a given hashtable.

- void pmath_ht_destroy (pmath_hashtable_t ht)

  Destroy a given hashtable.
- void pmath_ht_clear (pmath_hashtable_t ht)

  Clear a given hashtable.
- unsigned int pmath_ht_capacity (pmath_hashtable_t ht)

  Get the capacity of a given hashtable.
- unsigned int pmath_ht_count (pmath_hashtable_t ht)

  Get the number of valid entries in a given hashtable.
- void ∗ pmath_ht_entry (pmath_hashtable_t ht, unsigned int i)

  Get any entry of a given hashtable.
- void ∗ pmath_ht_search (pmath_hashtable_t ht, void ∗key)

  Search for an entry in a given hashtable.
- void ∗ pmath_ht_insert (pmath_hashtable_t ht, void ∗entry)

  Insert an entry into a given hashtable.
- void ∗ pmath_ht_remove (pmath_hashtable_t ht, void ∗key)

  Remove an entry from a given hashtable.

9.15.1 Detailed Description

A general hashtable implementation.

The user of this API is responsible for the memory layout of the entries.

9.15.2 Typedef Documentation

9.15.2.1 typedef struct _pmath_hashtable_t∗ pmath_hashtable_t

9.15.2.2 typedef void(∗ pmath_ht_entry_callback_t)(void ∗entry, void ∗data)

A callback function for hashtable entries.

Parameters:

entry An entry. Never PMATH_NULL.

data Additional data for the callback.

9.15.2.3 typedef void∗(∗ pmath_ht_entry_copy_t)(void ∗entry)

An entry copy function.

Parameters:

entry An entry. Never PMATH_NULL.

Returns:

A new entry that is a copy.

9.15.2.4 typedef pmath_bool_t(∗ pmath_ht_entry_equal_func_t)(void ∗entry1, void ∗entry2)

An entry comparision function.

Parameters:

entry1 The first entry.

entry2 The second entry.

Returns:

TRUE if both enties' keys are equal, FALSE otherwise.

9.15.2.5 typedef pmath_bool_t(∗ pmath_ht_entry_equals_key_func_t)(void ∗entry, void ∗key)

An entry to key comparision function.

Parameters:

entry An entry.

key The key of another entry.

Returns:

TRUE if both entry's key equals the given key, FALSE otherwise.

9.15.2.6 typedef unsigned int(∗ pmath_ht_entry_hash_func_t)(void ∗entry)

An entry hash function.

Parameters:

entry An entry. Never PMATH_NULL.

Returns:

A hash value for the entry's key.

9.15.2.7 typedef unsigned int(∗ pmath_ht_key_hash_func_t)(void ∗key)

A key hash function.

Parameters:

key A key.

Returns:

A hash value for the key.

9.15.3 Function Documentation

9.15.3.1 unsigned int pmath_ht_capacity (pmath_hashtable_t *ht*) `[inherited]`

Get the capacity of a given hashtable.

Parameters:

ht The hashtable.

Returns:

The current maximum possible index of an entry in the table.

9.15.3.2 void pmath_ht_clear (pmath_hashtable_t *ht*) `[inherited]`

Clear a given hashtable.

Parameters:

ht The hashtable.

9.15.3.3 pmath_hashtable_t pmath_ht_copy (pmath_hashtable_t *ht*, pmath_ht_entry_copy_t *entry_copy*) `[inherited]`

Copy a given hashtable.

Parameters:

ht The old hashtable

entry_copy A function for copying entires.

Returns:

The new hashtable or NULL on error.

9.15.3.4 unsigned int pmath_ht_count (pmath_hashtable_t *ht*) `[inherited]`

Get the number of valid entries in a given hashtable.

Parameters:

ht The hashtable.

Returns:

The number of non-NULL entires.

9.15.3.5 pmath_hashtable_t pmath_ht_create (const pmath_ht_class_t ∗ *klass*, unsigned int *minsize*) `[inherited]`

Create a new hashtable.

Parameters:

klass An interface pointer.

minsize Initial minimal size.

Returns:

The new hashtable or NULL on error.

9.15.3.6 void pmath_ht_destroy (pmath_hashtable_t *ht*) [inherited]

Destroy a given hashtable.

Parameters:

ht The hashtable.

9.15.3.7 void ∗ pmath_ht_entry (pmath_hashtable_t *ht*, unsigned int *i*) [inherited]

Get any entry of a given hashtable.

Parameters:

ht The hashtable.

i The index. from range 0..pmath_ht_capacity(ht) - 1

Returns:

The entry or NULL. It is owned by the table.

9.15.3.8 void ∗ pmath_ht_insert (pmath_hashtable_t *ht*, void ∗ *entry*) [inherited]

Insert an entry into a given hashtable.

Parameters:

ht The hashtable.

entry The entry. It must be compatible with all functions of the table's interface.

Returns:

NULL or a possible old entry or the entry itself in case of an error. You must destroy it.

9.15.3.9 void ∗ pmath_ht_remove (pmath_hashtable_t *ht*, void ∗ *key*) [inherited]

Remove an entry from a given hashtable.

Parameters:

ht The hashtable.

key The entry's key. It will be send to the kes_hash and entry_keys_equal functions of the table's interface pointer.

Returns:

NULL or the old entry. You must destroy it.

9.15.3.10 void ∗ pmath_ht_search (pmath_hashtable_t *ht*, void ∗ *key*) `[inherited]`

Search for an entry in a given hashtable.

Parameters:

ht The hashtable.

key The key. It will be send to the kes_hash and entry_keys_equal functions of the table's interface pointer.

Returns:

The entry or NULL. It is owned by the table.

9.16 Object Utility Functions

Utility functuions for pMath Objects and Expressions.

Typedefs

- typedef pmath_bool_t(∗ pmath_stack_walker_t)(pmath_t head, void ∗closure)

  A stack walker function.

Functions

- pmath_bool_t pmath_is_expr_of (pmath_t obj, pmath_symbol_t head)

  Check if an object is an expression with a specified head.

- pmath_bool_t pmath_is_expr_of_len (pmath_t obj, pmath_symbol_t head, size_t length)

  Check if an object is an expression with a specified head and length.

- pmath_t pmath_current_head (void)

Get the currently evaluated function.

- void pmath_walk_stack (pmath_stack_walker_t walker, void ∗closure)

  Walk up the current thread's and its parents' stack.

- void pmath_gather_begin (pmath_t pattern)

  Start gathering emitted objects.

- pmath_expr_t pmath_gather_end (void)

  Finish gathering emitted objects.

- void pmath_emit (pmath_t object, pmath_t tag)

  Emit an object to be gathered by the appropriate surounding pmath_gather_begin() ... pmath_gather_end() function pair.

- pmath_t pmath_build_value_v (const char ∗format, va_list args)

  Generate a List of objects with a format string.

- pmath_t pmath_build_value (const char ∗format,...)

  Generate a List of objects with a format string.

- pmath_expr_t pmath_options_extract (pmath_expr_t expr, size_t last_nonoption)

  Extract option values from an expression.

- pmath_t pmath_option_value (pmath_t fn, pmath_t name, pmath_t extra)

  Retrieve a option value of a given function.

9.16.1 Detailed Description

Utility functuions for pMath Objects and Expressions.

Here are some utility functions that simplify access to Expressions (or pMath Objects in general), but do not realy fit one of these topics.

9.16.2 Typedef Documentation

9.16.2.1 typedef pmath_bool_t(∗ pmath_stack_walker_t)(pmath_t head, void ∗closure)

A stack walker function.

The return value specifies, whether the walk on the stack go on.

9.16.3 Function Documentation

9.16.3.1 pmath_t pmath_build_value (const char * *format*, ...) `[related, inherited]`

Generate a List of objects with a format string.

Parameters:

format A string that specifies the tuple's item's type. See below.

... The tuple/list items

The format string characters specify the item's type:

- `b` [int] Converts a C int to True or False.

- `i` [int]
- `l` [long int]
- `k` [long long int]
- `n` [ssize_t]

- `I` [unsigned int]
- `L` [unsigned long int]
- `K` [unsigned long long int]
- `N` [size_t]

- `f` [double] NaN's and Infinity are converted to the symbols Undefined and +/- Infinity respectively.

- `o` [pmath_t] A pMath object, the reference is stolen.

- `c` [int] Convert a C int representing a (unicode) character to a string of length 1.

- `s` [char*] Converts a zero-terminated C string to a pMath string using Latin-1 encoding.

- `s#` [char*,int] Takes a char buffer and a length to build a pMath string of that length using Latin-1 encoding.

- `z` [char*] Takes a zero-terminated C string and converts it to a symbol using pmath_symbol_find().

- `u` [char∗] Converts a zero-terminated C string to a pMath string using UTF-8 encoding.

- `u#` [char∗,int] Takes a char buffer and a length to build a pMath string of that length using UTF-8 encoding.

- `U` [uint16_t∗] Converts a zero-terminated double-byte C string to a pMath string using UTF-16 encoding. This is generally useful only where sizeof(uint16_t) == sizeof(wchar_t), e.g. on Windows but not on Linux.

- `U#` [uint16_t∗,int] Takes a character buffer and a length to build a pMath string of that length using UTF-16 encoding. This is generally useful only on platforms with sizeof(uint16_t) == sizeof(wchar_t), e.g. on Windows but not on Linux.

- `Ctt` [matching the 2 *`t`*'s] Build a complex value.
- `Qtt` [matching the 2 *`t`*'s] Build a rational value.

- `(`*`items`*`)` [matching *`items`*] constructs a sublist of items.

Note:

When the format string denotes only one object, this object will be returned alone. So for a pmath_t x, pmath_build_value("o", x) == x.
If you want to return a list in any case, use "(...)": "i" gives an integer, "ii" and "(ii)" a list of two integers and "(i)" a list of one integer.

9.16.3.2 pmath_t pmath_build_value_v (const char ∗ *format*, va_list *args*) `[related, inherited]`

Generate a List of objects with a format string.

Parameters:

format A string that specifies the tuple's item's type.

args A va_list - variable argument list.

See also:

pmath_build_value

9.16.3.3 pmath_t pmath_current_head (void)

Get the currently evaluated function.

Returns:

The head of the expression that is currently evaluated (in the calling thread). You have to destroy it.

9.16.3.4 void pmath_emit (pmath_t *object*, pmath_t *tag*) `[related, inherited]`

Emit an object to be gathered by the appropriate surounding pmath_gather_begin() ... pmath_gather_end() function pair.

Parameters:

object The objcet to be emitted, it will be freed.

tag A tag object. The sourounding Gather() with a pattern, that matches `tag` will collect the `object`. `tag` will be freed.

See also:

pmath_gather_begin
pmath_gather_end

9.16.3.5 void pmath_gather_begin (pmath_t *pattern*) `[related, inherited]`

Start gathering emitted objects.

Parameters:

pattern A pattern that is used to determine which emitted objects should be gathered (testing the emit-tag, not the object itself). It will be freed.

Use pmath_gather_end() to finish gathering. Calls to pmath_gather_begin() ... pmath_gather_end() can be nested.

The emit-and-gather mechanism is useful when you want to create a list but do not know its final length in advance.

See also:

pmath_emit

9.16.3.6 pmath_expr_t pmath_gather_end (void) `[related, inherited]`

Finish gathering emitted objects.

Returns:

A list of all emitted objects since the last pmath_gather_begin() whose emit-tag matched the `pattern` parameter given to that pmath_gather_begin(). You must free it.

See also:

pmath_emit

9.16.3.7 pmath_bool_t pmath_is_expr_of (pmath_t *obj*, pmath_symbol_t *head*)

Check if an object is an expression with a specified head.

Parameters:

obj A pMath object. It wont be freed.

head A pMath symbol. It wont be freed.

Returns:

TRUE iff obj is an expression with the given *head* symbol.

9.16.3.8 pmath_bool_t pmath_is_expr_of_len (pmath_t *obj*, pmath_symbol_t *head*, size_t *length*)

Check if an object is an expression with a specified head and length.

Parameters:

obj A pMath object. It wont be freed.

head A pMath symbol. It wont be freed.

length The requested expression length.

Returns:

TRUE iff obj is an expression with the given *length* and *head* symbol.

9.16.3.9 pmath_t pmath_option_value (pmath_t *fn*, pmath_t *name*, pmath_t *extra*) `[related, inherited]`

Retrieve a option value of a given function.

Parameters:

fn The function for which the requested option value is defined. It wont be freed. If it is PMATH_NULL, the current head (see pmath_current_head) will be used.

name The name of the option value (in general, a symbol). It wont be freed.

extra A list of extra option rules or PMATH_UNDEFINED. It wont be freed. If it is not PMATH_UNDEFINED, it must be a rule (‘a->b‘, ‘a:>b‘) or a list of rules.

Returns:

The requested option value.

9.16.3.10 pmath_expr_t pmath_options_extract (pmath_expr_t *expr*, size_t *last_nonoption*) `[related, inherited]`

Extract option values from an expression.

Parameters:

expr The expression containing option values. It wont be freed.

last_nonoption The index of the last argument that is not an option rule.

Returns:

A list of all given option values or PMATH_NULL on error. You must destroy it.

Imagine, `expr` = ‘f(a,b,A->1,B->2)‘ and `last_nonoption` = 2, then the return value is a list ‘{A->1, B->2}‘. You can now use this return value as the `extra` parameter in pmath_option_value().

If `last_nonoption` was 1, a message would be generated (b is no rule ...) and the return value is PMATH_NULL. In that case, the calling function should have no further effects and return.

9.16.3.11 void pmath_walk_stack (pmath_stack_walker_t *walker*, void * *closure*)

Walk up the current thread’s and its parents’ stack.

Parameters:

walker A callback function.

closure A pointer that will be provided to walker as the second argument.

9.17 Memory Management

Memory management for pMath.

Functions

- void ∗ pmath_mem_alloc (size_t size)

  Allocate some amount of memory.

- void ∗ pmath_mem_realloc (void ∗p, size_t new_size)

  Change the size of a memory-chunk.

- void ∗ pmath_mem_realloc_no_failfree (void ∗p, size_t new_size)

  Change the size of a memory-chunk.

- void pmath_mem_free (void ∗p)

  Free a memory-chunk.

- void pmath_mem_usage (size_t ∗current, size_t ∗max)

  Get memory usage information.

9.17.1 Detailed Description

Memory management for pMath.

These functions may return NULL. In this case, the current evaluation will abort and used cache will be freed to safe memory (the garbage collector is invoked and a pMath exception is thrown so pmath_aborting() yields TRUE).

9.17.2 Function Documentation

9.17.2.1 void∗ pmath_mem_alloc (size_t *size*)

Allocate some amount of memory.

Parameters:

size The number of bytes to be allocated.

Returns:

A pointer to a block of mamory of at least size bytes or NULL.

You must free the result with pmath_mem_free() or indirectly via pmath_mem_realloc().

9.17.2.2 void pmath_mem_free (void ∗ *p*)

Free a memory-chunk.

Parameters:

p NULL or a pointer to a block of old_size bytes allocated with pmath_mem_alloc() or pmath_mem_realloc().

9.17.2.3 void∗ pmath_mem_realloc (void ∗ *p*, size_t *new_size*)

Change the size of a memory-chunk.

Parameters:

p NULL or a pointer to a block of old_size bytes allocated with pmath_mem_alloc() or pmath_mem_realloc().

new_size The requested new size.

Returns:

A pointer to a block of at least new_size bytes or NULL.

If there is not enough memory available or if new_size == 0, NULL is returned. Otherwise, the result points to a block of new_size bytes, whose first min(old_size,new_size) bytes are copied from the old p. The rest is initialized with 0.

The old pointer p will _allways_be freed. even, if the resizing failed with NULL.

You must later free the result with pmath_mem_free() or indirectly via pmath_mem_realloc().

9.17.2.4 void∗ pmath_mem_realloc_no_failfree (void ∗ *p*, size_t *new_size*)

Change the size of a memory-chunk.

Parameters:

p NULL or a pointer to a block of old_size bytes allocated with pmath_mem_alloc() or pmath_mem_realloc().

new_size The requested new size.

Returns:

A pointer to a block of at least new_size bytes or NULL.

If there is enough memory, this acts like pmath_mem_realloc(). Otherwise, p is *not* freed and NULL is returned.

9.17.2.5 void pmath_mem_usage (size_t ∗ *current*, size_t ∗ *max*)

Get memory usage information.

Parameters:

current Here goes the number of currently allocated bytes.

max here goes the maximum number of allocated bytes so far.

9.18 Messages

Error handling and informing the user.

Functions

- void pmath_message (pmath_symbol_t symbol, const char ∗tag, size_t argcount,...)

  Print a message with pMath object arguments.

- void pmath_message_argxxx (size_t given, size_t min, size_t max)

  Generate a General::arg∗ message (invalid argument count).

- pmath_string_t pmath_message_find_text (pmath_t name)

  Find a message's text.

- void pmath_message_syntax_error (pmath_string_t code, int position, pmath_string_t filename, int lines_before_code)

  Print a syntax warning or error message.

9.18.1 Detailed Description

Error handling and informing the user.

When you encounter an error such as wrong argument usage in pMath functions, you just print out a message and return the handled expression unevaluated. You do *not* call pmath_abort_please().

Example: The built-in Sin function invokes Sin::arg1 and returns unevaluated when it is called with a wrong number of arguments.

If you notice that a memory allocation failed, do nothing at all (even do not try to print a message). pMath automatically calls pmath_abort_please() on out-of-memory and thus no message would be shown.

Messages are similar to Mathematicas approach. On the language level, you enter 'Message(symbol::tag, ...)' to print a message with optional inserted values '...'. You can use Backquotes to refer to given values.

You can use all messages of the symbol General with every other symbol.

9.18.2 Function Documentation

9.18.2.1 void pmath_message (pmath_symbol_t *symbol*, const char ∗ *tag*, size_t *argcount*, ...)

Print a message with pMath object arguments.

Parameters:

symbol The symbol that defines the message. It wont be freed. PMATH_NULL is be treated as pmath_current_head().

tag The message's tag as a zero-terminated C string.

argcount The number of following arguments.

... Exactly *argcount* pMath objects. They will all be freed.

Note:

If symbol==PMATH_NULL and pmath_current_head() is an expression f(), the function acts as if pmath_current_head() was f.
All the symbols and expressions in ... will be embedded in HoldForm(...), because Message() would evaluate them. If you want one of the values to be evaluated, do it manually.

9.18.2.2 void pmath_message_argxxx (size_t *given*, size_t *min*, size_t *max*)

Generate a General::arg∗ message (invalid argument count).

Parameters:

given The given number of arguments.

min The minimum expected number of arguments.

max The maximum expected number of arguments.

One of the following messages may be generated: General::arg1

General::argxu

General::argx

General::argtu

General::argt

General::argru

General::argr

General::argmu

General::argm

9.18.2.3 pmath_string_t pmath_message_find_text (pmath_t *name*)

Find a message's text.

Parameters:

> ***name*** An expression of the form symbol::name (that is MessageName(symbol, "name")). It will be freed.

Returns:

> The message's content or PMATH_NULL if nothing was found or the magic value PMATH_UNDEFINED, if *name* does not have the expected form.

This function can be used in front-ends that overwrite the built-in Message function.

9.18.2.4 void pmath_message_syntax_error (pmath_string_t *code*, int *position*, pmath_string_t *filename*, int *lines_before_code*)

Print a syntax warning or error message.

Parameters:

> ***code*** The code. A pMath string. It wont be freed.
>
> ***position*** The position of the syntax error in the code.
>
> ***filename*** The file from where the input came or PMATH_NULL to omit it. It will be freed.
>
> ***lines_before_code*** The number of lines in the input file before *code* was read. This is ignored if *filename* is PMATH_NULL.

This function produces a Syntax::bgn, Syntax::bgnf, Syntax::nxt, Syntax::nxtf, Syntax::more, Syntax::moref, Syntax::newl or Syntax::newlf message, depending on where the syntax error is in the *code* and whether *filename* is PMATH_NULL or not. It can be used to report errors/warnings from pmath_spans_from_string().

9.19 Threadsafe Stacks

Fast threadsafe stacks in pMath.

Data Structures

- class pmath_stack_t

  The type of pMath's threadsafe stacks.

Typedefs

- typedef struct _pmath_stack_t ∗ pmath_stack_t

Functions

- pmath_stack_t pmath_stack_new (void)

  Create an empty stack.

- void pmath_stack_free (pmath_stack_t stack)

  Destroy a stack.

- void pmath_stack_push (pmath_stack_t stack, void ∗item)

  Push an item onto a stack.

- void ∗ pmath_stack_pop (pmath_stack_t stack)

  Pop an item from a stack.

9.19.1 Detailed Description

Fast threadsafe stacks in pMath.

pmath_stack_t is a stack abstraction that provides threadsafe push and pop operations. You can push and pop any structure whose first element is a pointer (which you must not touch).

If your CPU supports it, very fast lockfree operations are used for push and pop.

References

- Fober, Orlarey, Letz: "Lock-Free Techniques for Concurrent Access to Shared Objects" (`http://jim.afim-asso.org/jim2002/articles/L17_Fober.pdf`)

9.19.2 Typedef Documentation

9.19.2.1 typedef struct _pmath_stack_t∗ pmath_stack_t

9.19.3 Function Documentation

9.19.3.1 void pmath_stack_free (pmath_stack_t *stack*)

Destroy a stack.

Parameters:

stack A stack previously created with pmath_stack_new(). May be PMATH_NULL.

You must manually pop and free all items on the stack before calling this function, because those items would not be freed automatically.

9.19.3.2 pmath_stack_t pmath_stack_new (void)

Create an empty stack.

Returns:

A new stack or PMATH_NULL.

Note that you cannot push anything onto a PMATH_NULL stack, so check the result. Free the result with pmath_stack_free().

9.19.3.3 void∗ pmath_stack_pop (pmath_stack_t *stack*)

Pop an item from a stack.

Parameters:

stack The stack to pop an item from. Must not be PMATH_NULL.

Returns:

The top of stack or PMATH_NULL if it is empty.

9.19.3.4 void pmath_stack_push (pmath_stack_t *stack*, void $*$ *item*)

Push an item onto a stack.

Parameters:

stack The stack to where the item should be pushed. Must not be PMATH_NULL.

item The item to be pushed. It must point to a structure whose first element is a pointer. Must not be PMATH_NULL.

9.20 Version Information

Functions

- void pmath_version_datetime (int $*$year, int $*$month, int $*$day, int $*$hour, int $*$minute, int $*$second)

  Get the date and time when pMath was compiled.

- double pmath_version_number (void)

  Get version number (major + minor/100).

- long pmath_version_number_part (int index)

  Get version number part.

9.20.1 Function Documentation

9.20.1.1 void pmath_version_datetime (int $*$ *year*, int $*$ *month*, int $*$ *day*, int $*$ *hour*, int $*$ *minute*, int $*$ *second*)

Get the date and time when pMath was compiled.

9.20.1.2 double pmath_version_number (void)

Get version number (major + minor/100).

9.20.1.3 long pmath_version_number_part (int *index*)

Get version number part.

Parameters:

index The number index. Major=1, Minor=2, Revision=3, Subversion=4

9.21 Front-ends

Functions for use front-ends.

Functions

- pmath_bool_t pmath_continue_after_abort (void)

  Requests pMath to stop aborting the current evaluation.

- pmath_t pmath_session_execute (pmath_t input, pmath_bool_t ∗aborted)

  Execute an expression and change $History and $Line appropriately.

- pmath_t pmath_session_start (void)

  Saves some global state when an interactive dialog session starts.

- void pmath_session_end (pmath_t old_state)

  Restore some global state when an interactive dialog session ends.

- pmath_bool_t pmath_init (void)

  Initialize the pMath CAS library.

- void pmath_done (void)

  Free all resources used by the pMath CAS library and unload all modules.

9.21.1 Detailed Description

Functions for use front-ends.

A front-end to pMath is an executable that initializes and finalizes the library and handles User input and output or otherwise invokes expression evalueation with the library. That is, what pMath does: parse and evaluate pMath code.

9.21.2 Function Documentation

9.21.2.1 pmath_bool_t pmath_continue_after_abort (void)

Requests pMath to stop aborting the current evaluation.

Returns:

Whether the global aborting-flag was set (by pmath_abort_please())

This is for use in front-ends to allow the user to continue working after he aborted an evaluation.

Any uncought exception will be deleted silently.

This function also clears the $MessageCount cache.

See also:

pmath_abort_please()

9.21.2.2 void pmath_done (void)

Free all resources used by the pMath CAS library and unload all modules.

See also:

pmath_init

9.21.2.3 pmath_bool_t pmath_init (void)

Initialize the pMath CAS library.

Returns:

TRUE, if the initialization was successfull, FALSE otherwise.

Any thread must call pmath_init() before using any other pMath function (exception: Atomic Operations). If the initialization fails, the thread should stop. Otherwise, pmath_done() must be called before the thread ends.

When this function is called for the first time, the `maininit.pmath` file is executed. This file is searched in the following directories in order:

1. The application directory
2. The directory specified in the `PMATH_BASEDIRECTORY` environment variable
3. Some operating system specific directories:
   - /etc/pmath, /etc/local/pmath, /usr/share/pmath, /usr/share/local/pmath on Unix like systems
   - The "pmath" subfolder in the "common program files" and "program files" directories on Microsoft Windows, if existent.

9.21.2.4 void pmath_session_end (pmath_t *old_state*)

Restore some global state when an interactive dialog session ends.

Parameters:

old_state The object returned by pmath_session_start(). It will be freed.

This function should be used by a frontend when implementing the Dialog() function.

9.21.2.5 pmath_t pmath_session_execute (pmath_t *input*, pmath_bool_t ∗ *aborted*)

Execute an expression and change $History and $Line appropriately.

Parameters:

input The input expression. It will be freed.

aborted Optional address of a flag that will be set iff the evaluation was aborted by pmath_abort_please()

Returns:

The return value is the same as pmath_evaluate(input)

This function also calls pmath_collect_temporary_symbols before evaluation.

9.21.2.6 pmath_t pmath_session_start (void)

Saves some global state when an interactive dialog session starts.

Returns:

A pMath object to be given to pmath_session_end()

This function should be used by a frontend when implementing the Dialog() function.

10 Namespace Documentation

10.1 pmath Namespace Reference

Provides the C++ binding.

Data Structures

- class Expr

  A wrapper for pmath_t and drived types.

- class String

  A wrapper for pmath_string_t.

- class Gather

  Utility class for emitting and gathering expressions/building lists.

- class File

  A wrapper for pMath file objects (data streams).

- class BinaryFile

  A wrapper for pMath binary file objects (byte data streams).

- class ReadableBinaryFile

  A wrapper for readable pMath binary file objects (byte data streams).

- class WriteableBinaryFile

  A wrapper for writeable pMath binary file objects (byte data streams).

- class TextFile

  A wrapper for pMath text file objects (byte data streams).

- class ReadableTextFile

  A wrapper for pMath readable text file objects (byte data streams).

- class WriteableTextFile

  A wrapper for pMath writeable text file objects (byte data streams).

- class UserStream

  Abstract base class for C++ callbacks used as pMath files.

- class BinaryUserStream

  Abstract base class for C++ callbacks used as pMath binary files.

- class TextUserStream

  Abstract base class for C++ callbacks used as pMath text files.

Functions

- Expr Number (double d)
- Expr Complex (const Expr &re, const Expr &im)
- Expr Imaginary (const Expr &im)
- Expr Rational (const Expr &num, const Expr &den)
- Expr Ref (pmath_t o)
- Expr Symbol (pmath_symbol_t h)
- Expr SymbolPi ()
- Expr MakeList (size_t len)
- Expr Call (const Expr &h)
- Expr Call (const Expr &h, const Expr &x1)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8, const Expr &x9)
- Expr List ()
- Expr List (const Expr &x1)
- Expr List (const Expr &x1, const Expr &x2)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8, const Expr &x9)
- Expr Rule (const Expr &l, const Expr &r)
- Expr RuleDelayed (const Expr &l, const Expr &r)
- Expr Power (const Expr &x, const Expr &y)

- Expr Sqrt (const Expr &x)
- Expr Inv (const Expr &x)
- Expr Exp (const Expr &x)
- Expr Log (const Expr &x)
- Expr Log (const Expr &b, const Expr &x)
- Expr Sin (const Expr &x)
- Expr Cos (const Expr &x)
- Expr Tan (const Expr &x)
- Expr ArcSin (const Expr &x)
- Expr ArcCos (const Expr &x)
- Expr ArcTan (const Expr &x)
- Expr Times (const Expr &x1, const Expr &x2)
- Expr Times (const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Times (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr Divide (const Expr &x, const Expr &y)
- Expr Plus (const Expr &x1, const Expr &x2)
- Expr Plus (const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Plus (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr Minus (const Expr &x)
- Expr Minus (const Expr &x, const Expr &y)
- Expr Abs (const Expr &x)
- Expr Arg (const Expr &x)
- Expr Sign (const Expr &x)
- Expr Re (const Expr &x)
- Expr Im (const Expr &x)
- Expr Ceiling (const Expr &x)
- Expr Ceiling (const Expr &x, const Expr &a)
- Expr Floor (const Expr &x)
- Expr Floor (const Expr &x, const Expr &a)
- Expr Round (const Expr &x)
- Expr Round (const Expr &x, const Expr &a)
- Expr Quotient (const Expr &m, const Expr &n)
- Expr Quotient (const Expr &m, const Expr &n, const Expr &d)
- Expr Mod (const Expr &m, const Expr &n)
- Expr Mod (const Expr &m, const Expr &n, const Expr &d)
- Expr Evaluate (const Expr &x)
- Expr ParseArgs (const char ∗code, const Expr &arglist)
- Expr Parse (const String &code)
- Expr Parse (const char ∗code)
- Expr Parse (const char ∗code, const Expr &x1)
- Expr Parse (const char ∗code, const Expr &x1, const Expr &x2)
- Expr Parse (const char ∗code, const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Parse (const char ∗code, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)

10.1.1 Detailed Description

Provides the C++ binding.

10.1.2 Function Documentation

10.1.2.1 Expr pmath::Abs (const Expr & *x*) `[inline]`

10.1.2.2 Expr pmath::ArcCos (const Expr & *x*) `[inline]`

10.1.2.3 Expr pmath::ArcSin (const Expr & *x*) `[inline]`

10.1.2.4 Expr pmath::ArcTan (const Expr & *x*) `[inline]`

10.1.2.5 Expr pmath::Arg (const Expr & *x*) `[inline]`

10.1.2.6 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*, const Expr & *x7*, const Expr & *x8*, const Expr & *x9*) `[inline]`

10.1.2.7 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*, const Expr & *x7*, const Expr & *x8*) `[inline]`

10.1.2.8 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*, const Expr & *x7*) `[inline]`

10.1.2.9 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*) `[inline]`

10.1.2.10 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*) `[inline]`

10.1.2.11 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*) `[inline]`

10.1.2.12 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*) `[inline]`

10.1.2.13 Expr pmath::Call (const Expr & *h*, const Expr & *x1*, const Expr & *x2*) `[inline]`

10.1.2.14 Expr pmath::Call (const Expr & *h*, const Expr & *x1*) `[inline]`

10.1.2.15 Expr pmath::Call (const Expr & *h*) `[inline]`

10.1.2.16 Expr pmath::Ceiling (const Expr & *x*, const Expr & *a*) `[inline]`

10.1.2.17 Expr pmath::Ceiling (const Expr & *x*) `[inline]`

10.1.2.18 Expr pmath::Complex (const Expr & *re*, const Expr & *im*) `[inline]`

10.1.2.19 Expr pmath::Cos (const Expr & *x*) `[inline]`

10.1.2.20 Expr pmath::Divide (const Expr & *x*, const Expr & *y*) `[inline]`

10.1.2.21 Expr pmath::Evaluate (const Expr & *x*) `[inline]`

10.1.2.22 Expr pmath::Exp (const Expr & *x*) `[inline]`

10.1.2.23 Expr pmath::Floor (const Expr & *x*, const Expr & *a*) `[inline]`

10.1.2.24 Expr pmath::Floor (const Expr & *x*) `[inline]`

10.1.2.25 Expr pmath::Im (const Expr & *x*) `[inline]`

10.1.2.26 Expr pmath::Imaginary (const Expr & *im*) `[inline]`

10.1.2.27 Expr pmath::Inv (const Expr & *x*) `[inline]`

10.1.2.28 Expr pmath::List (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*, const Expr & *x7*, const Expr & *x8*, const Expr & *x9*) `[inline]`

10.1.2.29 Expr pmath::List (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*, const Expr & *x7*, const Expr & *x8*) `[inline]`

10.1.2.30 Expr pmath::List (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*, const Expr & *x7*) `[inline]`

10.1.2.31 Expr pmath::List (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*, const Expr & *x6*) `[inline]`

10.1.2.32 Expr pmath::List (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*, const Expr & *x5*) `[inline]`

10.1.2.33 Expr pmath::List (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*) `[inline]`

10.1.2.34 Expr pmath::List (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*) `[inline]`

10.1.2.35 Expr pmath::List (const Expr & *x1*, const Expr & *x2*) `[inline]`

10.1.2.36 Expr pmath::List (const Expr & *x1*) `[inline]`

10.1.2.37 Expr pmath::List () `[inline]`

10.1.2.38 Expr pmath::Log (const Expr & *b*, const Expr & *x*) `[inline]`

10.1.2.39 Expr pmath::Log (const Expr & *x*) `[inline]`

10.1.2.40 Expr pmath::MakeList (size_t *len*) `[inline]`

10.1.2.41 Expr pmath::Minus (const Expr & *x*, const Expr & *y*) `[inline]`

10.1.2.42 Expr pmath::Minus (const Expr & *x*) `[inline]`

10.1.2.43 Expr pmath::Mod (const Expr & *m*, const Expr & *n*, const Expr & *d*) `[inline]`

10.1.2.44 Expr pmath::Mod (const Expr & *m*, const Expr & *n*) `[inline]`

10.1.2.45 Expr pmath::Number (double *d*) `[inline]`

10.1.2.46 Expr pmath::Parse (const char ∗ *code*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*) `[inline]`

10.1.2.47 Expr pmath::Parse (const char ∗ *code*, const Expr & *x1*, const Expr & *x2*, const Expr & *x3*) `[inline]`

10.1.2.48 Expr pmath::Parse (const char ∗ *code*, const Expr & *x1*, const Expr & *x2*) `[inline]`

10.1.2.49 Expr pmath::Parse (const char ∗ *code*, const Expr & *x1*) `[inline]`

10.1.2.50 Expr pmath::Parse (const char ∗ *code*) `[inline]`

10.1.2.51 Expr pmath::Parse (const String & *code*) `[inline]`

10.1.2.52 Expr pmath::ParseArgs (const char ∗ *code*, const Expr & *arglist*) `[inline]`

10.1.2.53 Expr pmath::Plus (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*) `[inline]`

10.1.2.54 Expr pmath::Plus (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*) `[inline]`

10.1.2.55 Expr pmath::Plus (const Expr & *x1*, const Expr & *x2*) `[inline]`

10.1.2.56 Expr pmath::Power (const Expr & *x*, const Expr & *y*) `[inline]`

10.1.2.57 Expr pmath::Quotient (const Expr & *m*, const Expr & *n*, const Expr & *d*) `[inline]`

10.1.2.58 Expr pmath::Quotient (const Expr & *m*, const Expr & *n*) `[inline]`

10.1.2.59 Expr pmath::Rational (const Expr & *num*, const Expr & *den*) `[inline]`

10.1.2.60 Expr pmath::Re (const Expr & *x*) `[inline]`

10.1.2.61 Expr pmath::Ref (pmath_t *o*) `[inline]`

10.1.2.62 Expr pmath::Round (const Expr & *x*, const Expr & *a*) `[inline]`

10.1.2.63 Expr pmath::Round (const Expr & *x*) `[inline]`

10.1.2.64 Expr pmath::Rule (const Expr & *l*, const Expr & *r*) `[inline]`

10.1.2.65 Expr pmath::RuleDelayed (const Expr & *l*, const Expr & *r*) `[inline]`

10.1.2.66 Expr pmath::Sign (const Expr & *x*) `[inline]`

10.1.2.67 Expr pmath::Sin (const Expr & *x*) `[inline]`

10.1.2.68 Expr pmath::Sqrt (const Expr & *x*) `[inline]`

10.1.2.69 Expr pmath::Symbol (pmath_symbol_t *h*) `[inline]`

10.1.2.70 Expr pmath::SymbolPi () `[inline]`

10.1.2.71 Expr pmath::Tan (const Expr & *x*) `[inline]`

10.1.2.72 Expr pmath::Times (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*, const Expr & *x4*) `[inline]`

10.1.2.73 Expr pmath::Times (const Expr & *x1*, const Expr & *x2*, const Expr & *x3*) `[inline]`

10.1.2.74 Expr pmath::Times (const Expr & *x1*, const Expr & *x2*) `[inline]`

11 Data Structure Documentation

11.1 BinaryFile Class Reference

A wrapper for pMath binary file objects (byte data streams).

Inherits pmath::File.

Inherited by ReadableBinaryFile, and WriteableBinaryFile.

Public Member Functions

- BinaryFile ()
- BinaryFile (pmath_t file_object) throw ()
- BinaryFile (const Expr &file_object) throw ()
- BinaryFile (const BinaryFile &src) throw ()
- size_t get_buffer_size () const throw ()

  Get the current buffer size in bytes.
- bool set_buffer_size (size_t size) throw ()

  See file_set_binbuffer().

Static Public Member Functions

- static BinaryFile create_buffer (size_t mincapacity) throw ()

  Create a memory buffer for as a double ended queue.

11.1.1 Detailed Description

A wrapper for pMath binary file objects (byte data streams).

11.1.2 Constructor & Destructor Documentation

11.1.2.1 BinaryFile () `[inline]`

11.1.2.2 BinaryFile (pmath_t *file_object*) throw () `[inline, explicit]`

11.1.2.3 BinaryFile (const Expr & *file_object*) throw () `[inline, explicit]`

11.1.2.4 BinaryFile (const BinaryFile & *src*) throw () `[inline]`

11.1.3 Member Function Documentation

11.1.3.1 static BinaryFile create_buffer (size_t *mincapacity*) throw () `[inline, static]`

Create a memory buffer for as a double ended queue.

Returns:

The binary file. Can be used as ReadableBinaryFile and as WriteableBinaryFile

11.1.3.2 size_t get_buffer_size () const throw () `[inline]`

Get the current buffer size in bytes.

11.1.3.3 bool set_buffer_size (size_t *size*) throw () `[inline]`

See file_set_binbuffer().

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.2 BinaryUserStream Class Reference

Abstract base class for C++ callbacks used as pMath binary files.

Inherits pmath::UserStream.

Protected Member Functions

- virtual pmath_files_status_t status ()=0

Called by pMath to check for end-of-file and other errors.

- virtual void flush ()

  Called by pMath to flush data to disk.

- virtual size_t read (void ∗buffer, size_t buffer_size)=0

  Called by pMath to read data.

- virtual size_t write (const void ∗buffer, size_t buffer_size)=0

  Called by pMath to write data.

- BinaryFile convert_to_file (bool readable, bool writeable)

  Convert to a binary file. pMath will take ownership of the C++ object.

- ReadableBinaryFile convert_to_file_readonly ()
- WriteableBinaryFile convert_to_file_writeonly ()

11.2.1 Detailed Description

Abstract base class for C++ callbacks used as pMath binary files.

11.2.2 Member Function Documentation

11.2.2.1 BinaryFile convert_to_file (bool *readable*, bool *writeable*) `[inline, protected]`

Convert to a binary file. pMath will take ownership of the C++ object.

Because pMath now owns the C++ object, you must not touch it directly after this call

11.2.2.2 ReadableBinaryFile convert_to_file_readonly () `[inline, protected]`

11.2.2.3 WriteableBinaryFile convert_to_file_writeonly () `[inline, protected]`

11.2.2.4 virtual void flush () `[inline, protected, virtual]`

Called by pMath to flush data to disk.

11.2.2.5 virtual size_t read (void ∗ *buffer*, size_t *buffer_size*) `[protected, pure virtual]`

Called by pMath to read data.

11.2.2.6 virtual pmath_files_status_t status () `[protected, pure virtual]`

Called by pMath to check for end-of-file and other errors.

11.2.2.7 virtual size_t write (const void ∗ *buffer*, size_t *buffer_size*) `[protected, pure virtual]`

Called by pMath to write data.

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.3 Expr Class Reference

A wrapper for pmath_t and drived types.

Inherited by File, and String.

Public Member Functions

- Expr () throw ()

  Initialize with PMATH_NULL.

- Expr (pmath_t obj) throw ()

  Construct form a pmath_t, that will be freed automatically with the Expr.

- Expr (const Expr &src) throw ()

  Copy an Expr, inrecementing the reference counter.

- Expr (int i) throw ()

  Construct from an int.

- Expr (size_t i) throw ()

  Construct from an size_t.

- Expr (double f) throw ()

  Construct from a double. May yield Infinity or Undefined (NaN) values.

- ∼Expr () throw ()

  Destructor. Frees the wrapped object.

- Expr & operator= (const Expr &src) throw ()

  Copy an Expr. Increments the new value's reference counter and frees the old one.

- bool is_custom () const throw ()
- bool is_double () const throw ()
- bool is_expr () const throw ()
- bool is_float () const throw ()
- bool is_integer () const throw ()
- bool is_int32 () const throw ()
- bool is_magic () const throw ()
- bool is_mpfloat () const throw ()
- bool is_null () const throw ()
- bool is_number () const throw ()
- bool is_pointer () const throw ()
- bool is_quotient () const throw ()
- bool is_rational () const throw ()
- bool is_string () const throw ()
- bool is_symbol () const throw ()
- bool is_pointer_of (pmath_type_t type) const throw ()
- bool is_evaluated () const throw ()
- bool is_rule () const throw ()
- unsigned int hash () const throw ()

  Get a hash value.

- bool operator== (const Expr &other) const throw ()

  Check for identity. The pMath === operator.

- bool operator!= (const Expr &other) const throw ()

  Check for non-identity. The pMath =!= operator.

- int compare (const Expr &other) const throw ()

  Compare with another Expr. See pmath_compare().

- bool operator< (const Expr &other) const throw ()

  Compare with another Expr. See pmath_compare().

- bool operator<= (const Expr &other) const throw ()

  Compare with another Expr. See pmath_compare().

- bool operator> (const Expr &other) const throw ()

Compare with another Expr. See pmath_compare().

- bool operator>= (const Expr &other) const throw ()

  Compare with another Expr. See pmath_compare().

- pmath_t release () throw ()

  Return the pmath_t and discard it. Caller must pmath_unref() it.

- const pmath_t get () const throw ()

  Get the pmath_t. Reference is held by the Expr object.

- bool is_valid () const throw ()

  Check for not holding the null pointer.

- size_t expr_length () const throw ()

  Length of the expression or 0 on error.

- Expr operator[ ] (size_t i) const throw ()

  Get the i-th argument of the expression.

- void set (size_t i, Expr e) throw ()

  Change the i-th argument of an expression.

- void set (size_t i, size_t j, Expr e) throw ()

  Change the (i,j)-th argument of a matrix.

- double to_double (double def=0.0) const throw ()

  Convert to a double.

- String to_string (pmath_write_options_t options=0) const throw ()

  Convert to a string. The pMath ToString function.

- void write_to_file (WriteableTextFile file, pmath_write_options_t options=0) const throw ()

  Write to a file/text stream.

- pmath_serialize_error_t serialize (WriteableBinaryFile file) const throw ()

  Serialize to a binary file/stream.

- bool operator== (pmath_t o1, const Expr &o2) throw()

  check for identity. The pMath === operator.

- bool operator!= (pmath_t o1, const Expr &o2) throw()

  check for non-identity. The pMath =!= operator.

- bool operator== (const Expr &o1, pmath_t o2) throw()
- bool operator!= (const Expr &o1, pmath_t o2) throw()

Static Public Member Functions

- static Expr deserialize (ReadableBinaryFile file, pmath_serialize_error_t ∗error) throw ()

  Deserialize an Expr from a binary file/stream.

Static Protected Member Functions

- static void write_to_string (void ∗user, const uint16_t ∗data, int len) throw ()

Protected Attributes

- pmath_t _obj

11.3.1 Detailed Description

A wrapper for pmath_t and drived types.

This class wraps a single pmath_t, so its sizeof(Expr) == sizeof(pmath_t).

11.3.2 Constructor & Destructor Documentation

11.3.2.1 Expr () throw () `[inline]`

Initialize with PMATH_NULL.

11.3.2.2 Expr (pmath_t *obj*) throw () `[inline, explicit]`

Construct form a pmath_t, that will be freed automatically with the Expr.

11.3.2.3 Expr (const Expr & *src*) throw () `[inline]`

Copy an Expr, inrecementing the reference counter.

11.3.2.4 Expr (int *i*) throw () `[inline]`

Construct from an int.

11.3.2.5 Expr (size_t *i*) throw () `[inline]`

Construct from an size_t.

11.3.2.6 Expr (double *f*) throw () `[inline]`

Construct from a double. May yield Infinity or Undefined (NaN) values.

11.3.2.7 ∼Expr () throw () `[inline]`

Destructor. Frees the wrapped object.

11.3.3 Member Function Documentation

11.3.3.1 int compare (const Expr & *other*) const throw () `[inline]`

Compare with another Expr. See pmath_compare().

11.3.3.2 Expr deserialize (ReadableBinaryFile *file*, pmath_serialize_error_t ∗ *error*) throw () `[inline, static]`

Deserialize an Expr from a binary file/stream.

Parameters:

file The binary file/stream. It must be writeable.

error An error number is stored here. May be NULL if not needed.

Returns:

The deserialized expression.

11.3.3.3 size_t expr_length () const throw () `[inline]`

Length of the expression or 0 on error.

11.3.3.4 const pmath_t get () const throw () [inline]

Get the pmath_t. Reference is held by the Expr object.

11.3.3.5 unsigned int hash () const throw () [inline]

Get a hash value.

11.3.3.6 bool is_custom () const throw () [inline]

11.3.3.7 bool is_double () const throw () [inline]

11.3.3.8 bool is_evaluated () const throw () [inline]

11.3.3.9 bool is_expr () const throw () [inline]

11.3.3.10 bool is_float () const throw () [inline]

11.3.3.11 bool is_int32 () const throw () [inline]

11.3.3.12 bool is_integer () const throw () [inline]

11.3.3.13 bool is_magic () const throw () [inline]

11.3.3.14 bool is_mpfloat () const throw () `[inline]`

11.3.3.15 bool is_null () const throw () `[inline]`

11.3.3.16 bool is_number () const throw () `[inline]`

11.3.3.17 bool is_pointer () const throw () `[inline]`

11.3.3.18 bool is_pointer_of (pmath_type_t *type*) const throw () `[inline]`

11.3.3.19 bool is_quotient () const throw () `[inline]`

11.3.3.20 bool is_rational () const throw () `[inline]`

11.3.3.21 bool is_rule () const throw () `[inline]`

11.3.3.22 bool is_string () const throw () `[inline]`

11.3.3.23 bool is_symbol () const throw () `[inline]`

11.3.3.24 bool is_valid () const throw () `[inline]`

Check for not holding the null pointer.

11.3.3.25 bool operator!= (const Expr & *o1*, pmath_t *o2*) throw() `[inline]`

11.3.3.26 bool operator!= (pmath_t *o1*, const Expr & *o2*) throw() `[inline]`

check for non-identity. The pMath =!= operator.

11.3.3.27 bool operator!= (const Expr & *other*) const throw () `[inline]`

Check for non-identity. The pMath =!= operator.

11.3.3.28 bool operator< (const Expr & *other*) const throw () `[inline]`

Compare with another Expr. See pmath_compare().

11.3.3.29 bool operator<= (const Expr & *other*) const throw () `[inline]`

Compare with another Expr. See pmath_compare().

11.3.3.30 Expr& operator= (const Expr & *src*) throw () `[inline]`

Copy an Expr. Increments the new value's reference counter and frees the old one.

11.3.3.31 bool operator== (const Expr & *o1*, pmath_t *o2*) throw() `[inline]`

11.3.3.32 bool operator== (pmath_t *o1*, const Expr & *o2*) throw() `[inline]`

check for identity. The pMath === operator.

11.3.3.33 bool operator== (const Expr & *other*) const throw () `[inline]`

Check for identity. The pMath === operator.

11.3.3.34 bool operator> (const Expr & *other*) const throw () `[inline]`

Compare with another Expr. See pmath_compare().

11.3.3.35 bool operator>= (const Expr & *other*) const throw () `[inline]`

Compare with another Expr. See pmath_compare().

11.3.3.36 Expr operator[ ] (size_t *i*) const throw () `[inline]`

Get the i-th argument of the expression.

Parameters:

i Index. May be > expr_length().

Returns:

The i-th argument if the object is a pmath_expr_t. expr[0] is the head, expr[1] the first argument and expr[length()] the last argument.

11.3.3.37 pmath_t release () throw () `[inline]`

Return the pmath_t and discard it. Caller must pmath_unref() it.

11.3.3.38 pmath_serialize_error_t serialize (WriteableBinaryFile *file*) const throw () `[inline]`

Serialize to a binary file/stream.

Parameters:

file The binary file/stream. It must be writeable.

Returns:

An error number.

11.3.3.39 void set (size_t *i*, size_t *j*, Expr *e*) throw () `[inline]`

Change the (i,j)-th argument of a matrix.

Parameters:

i The matrix row.

j The matrix column.

e The new element.

11.3.3.40 void set (size_t *i*, Expr *e*) throw () `[inline]`

Change the i-th argument of an expression.

Parameters:

i Index. May be > expr_length().

e The new element.

11.3.3.41 double to_double (double *def* = `0.0`) const throw () `[inline]`

Convert to a double.

Parameters:

def Optional default value.

Returns:

the double value if the object is a numeric object and *def* otherwise.

11.3.3.42 String to_string (pmath_write_options_t *options* = `0`) const throw () `[inline]`

Convert to a string. The pMath ToString function.

Parameters:

options Optional formating options.

Returns:

The String representation.

11.3.3.43 void write_to_file (WriteableTextFile *file*, pmath_write_options_t *options* = `0`) const throw () `[inline]`

Write to a file/text stream.

Parameters:

file The text file object. It must be writeable.

options Optional formating options.

11.3.3.44 static void write_to_string (void ∗ *user*, const uint16_t ∗ *data*, int *len*) throw () `[inline, static, protected]`

11.3.4 Field Documentation

11.3.4.1 pmath_t _obj `[protected]`

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.4 File Class Reference

A wrapper for pMath file objects (data streams).

Inherits pmath::Expr.

Inherited by BinaryFile, and TextFile.

Public Member Functions

- File () throw ()
- File (pmath_t file_object) throw ()
- File (const Expr &file_object) throw ()
- File (const File &src) throw ()
- bool has_capabilities (int properties) const throw ()

  Test for file properties/capabilities.

- bool is_file () const throw ()
- bool is_readable () const throw ()
- bool is_writeable () const throw ()
- bool is_binary () const throw ()
- bool is_text () const throw ()
- pmath_files_status_t status () const throw ()
- void flush () throw ()

  Flush the output buffer of a writeable file.

- void close () throw ()

  Closes a file immediatly instead of letting the garbage collector close it later.

11.4.1 Detailed Description

A wrapper for pMath file objects (data streams).

This class provides some stream utility functions in addition to Expr. Note that a pMath file does not have to correspond to any operating system file object.

11.4.2 Constructor & Destructor Documentation

11.4.2.1 File () throw () `[inline]`

11.4.2.2 File (pmath_t *file_object*) throw () `[inline, explicit]`

11.4.2.3 File (const Expr & *file_object*) throw () `[inline, explicit]`

11.4.2.4 File (const File & *src*) throw () `[inline]`

11.4.3 Member Function Documentation

11.4.3.1 void close () throw () `[inline]`

Closes a file immediatly instead of letting the garbage collector close it later.

11.4.3.2 void flush () throw () `[inline]`

Flush the output buffer of a writeable file.

11.4.3.3 bool has_capabilities (int *properties*) const throw () `[inline]`

Test for file properties/capabilities.

Parameters:

properties 0 or one or more of the PMATH_FILE_PROP_XXX constants.

Returns:

Whether the file has all the specified capabilities.

11.4.3.4 bool is_binary () const throw () `[inline]`

11.4.3.5 bool is_file () const throw () `[inline]`

11.4.3.6 bool is_readable () const throw () `[inline]`

11.4.3.7 bool is_text () const throw () `[inline]`

11.4.3.8 bool is_writeable () const throw () `[inline]`

11.4.3.9 pmath_files_status_t status () const throw () [inline]

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.5 Gather Class Reference

Utility class for emitting and gathering expressions/building lists.

Public Member Functions

- Gather () throw ()

  The constructor. Starts gathering.

- Gather (Expr pattern) throw ()
- ∼Gather () throw ()
- Expr end () throw ()

  end gathering. Calling end() multiple times returns PMATH_NULL.

Static Public Member Functions

- static void emit (Expr e) throw ()

  Emit a value to be gathered.

- static void emit (Expr e, Expr tag) throw ()

  Emit a value to be gathered.

Protected Attributes

- bool ended

11.5.1 Detailed Description

Utility class for emitting and gathering expressions/building lists.

Gathering begins with the construction of the object and ends with a call to end() or the object destruction. This removes the burden of calling pmath_gather_end() for every pmath_gather_begin().

11.5.2 Constructor & Destructor Documentation

11.5.2.1 Gather () throw () `[inline]`

The constructor. Starts gathering.

11.5.2.2 Gather (Expr *pattern*) throw () `[inline, explicit]`

11.5.2.3 ~Gather () throw () `[inline]`

11.5.3 Member Function Documentation

11.5.3.1 static void emit (Expr *e*, Expr *tag*) throw () `[inline, static]`

Emit a value to be gathered.

11.5.3.2 static void emit (Expr *e*) throw () `[inline, static]`

Emit a value to be gathered.

11.5.3.3 Expr end () throw () `[inline]`

end gathering. Calling end() multiple times returns PMATH_NULL.

11.5.4 Field Documentation

11.5.4.1 bool ended `[protected]`

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.6 pmath_atomic2_t Struct Reference

A 2-pointer-sized atomic variable type.

11.6.1 Detailed Description

A 2-pointer-sized atomic variable type.

The documentation for this struct was generated from the following file:

- pmath-util/concurrency/atomic.h

11.7 pmath_atomic_t Struct Reference

A pointer-sized atomic variable type.

11.7.1 Detailed Description

A pointer-sized atomic variable type.

Initialize it with PMATH_ATOMIC_STATIC_INIT.

The documentation for this struct was generated from the following file:

- pmath-util/concurrency/atomic.h

11.8 pmath_binary_file_api_t Class Reference

Access functions for binary files.

Data Fields

- size_t struct_size

  The structure size. Allways initialize this with `sizeof(pmath_binary_file_api_t)`.

- pmath_files_status_t(∗ status_function)(void ∗extra)

  A function to get the current file status This can be PMATH_NULL if read_function is also PMATH_NULL.

- size_t(∗ read_function)(void ∗extra, void ∗buffer, size_t buffer_size)

  An optional callback function for reading bytes.

- size_t(∗ write_function)(void ∗extra, const void ∗buffer, size_t buffer_size)

  An optional callback function for writing bytes.

- void(∗ flush_function)(void ∗extra)

  An optional callback function for flushing an output buffer.

11.8.1 Detailed Description

Access functions for binary files.

See also:

pmath_file_create_binary

11.8.2 Field Documentation

11.8.2.1 void(∗ flush_function)(void ∗extra)

An optional callback function for flushing an output buffer.

11.8.2.2 size_t(∗ read_function)(void ∗extra, void ∗buffer, size_t buffer_size)

An optional callback function for reading bytes.

11.8.2.3 pmath_files_status_t(∗ status_function)(void ∗extra)

A function to get the current file status This can be PMATH_NULL if read_function is also PMATH_NULL.

11.8.2.4 size_t struct_size

The structure size. Allways initialize this with `sizeof(pmath_binary_file_api_t)`.

11.8.2.5 size_t(∗ write_function)(void ∗extra, const void ∗buffer, size_t buffer_size)

An optional callback function for writing bytes.

The documentation for this class was generated from the following file:

- pmath-util/files.h

11.9 pmath_cstr_writer_info_t Struct Reference

Additional information for pmath_utf8_writer() or pmath_native_writer().

Data Fields

- void(∗ write_cstr)(void ∗user, const char ∗cstr)
- void ∗ user

11.9.1 Detailed Description

Additional information for pmath_utf8_writer() or pmath_native_writer().

11.9.2 Field Documentation

11.9.2.1 void∗ user

11.9.2.2 void(∗ write_cstr)(void ∗user, const char ∗cstr)

The documentation for this struct was generated from the following file:

- pmath-core/strings.h

11.10 pmath_custom_t Class Reference

The Custom Object class.

Inherits pmath_t.

Inherited by pmath_messages_t.

Public Member Functions

- pmath_custom_t pmath_custom_new (void ∗data, pmath_callback_t destructor)

  Create a custom object.

- void ∗ pmath_custom_get_data (pmath_custom_t custom)

  Get a custom object's data member.

- pmath_bool_t pmath_custom_has_destructor (pmath_custom_t custom, pmath_callback_t dtor)

  Check for a custom object's data type.

11.10.1 Detailed Description

The Custom Object class.

Since it is derived from pmath_t, you can provide it to any function that accepts a pmath_t.

The documentation for this class was generated from the following file:

- pmath-core/custom.h

11.11 pmath_expr_t Class Reference

The Expression class.

Inherits pmath_t.

Public Member Functions

- pmath_expr_t pmath_expr_new (pmath_t head, size_t length)

  Create a new expression.

- pmath_expr_t pmath_expr_new_extended (pmath_t head, size_t length,...)

  Create a new expression with all items given.

- pmath_expr_t pmath_expr_resize (pmath_expr_t expr, size_t new_length)

  Resize an expression.

- pmath_expr_t pmath_expr_append (pmath_expr_t expr, size_t count,...)

  Append some items to an expression.

- size_t pmath_expr_length (pmath_expr_t expr)

  Get an expression's length.

- pmath_t pmath_expr_get_item (pmath_expr_t expr, size_t index)

  Get an item from an expression.

- pmath_t pmath_expr_extract_item (pmath_expr_t expr, size_t index)

  Extract an item from an expression.

- pmath_expr_t pmath_expr_get_item_range (pmath_expr_t expr, size_t start, size_t length)

  Get multiple items from an expression.

- const pmath_t * pmath_expr_read_item_data (pmath_expr_t expr)

  Get a pointer to the expression's internal items array.

- pmath_expr_t pmath_expr_set_item (pmath_expr_t expr, size_t index, pmath_t item)

  Set an item in an expression.

- pmath_expr_t pmath_expr_remove_all (pmath_expr_t expr, pmath_t rem)

  Remove all occurencies of an object from an expression.

- pmath_expr_t pmath_expr_sort (pmath_expr_t expr)

  Sort an expression.

- pmath_expr_t pmath_expr_flatten (pmath_expr_t expr, pmath_t head, size_t depth)

  Flatten an expression.

Related Functions

(Note that these are not member functions.)

- void pmath_gather_begin (pmath_t pattern)

  Start gathering emitted objects.

- pmath_expr_t pmath_gather_end (void)

  Finish gathering emitted objects.

- void pmath_emit (pmath_t object, pmath_t tag)

  Emit an object to be gathered by the appropriate surounding pmath_gather_begin() ... pmath_gather_end() function pair.

- pmath_t pmath_evaluate_expression (pmath_expr_t expr, pmath_bool_t apply_rules)

  Partly evaluate an expression.

- pmath_expr_t pmath_options_extract (pmath_expr_t expr, size_t last_nonoption)

  Extract option values from an expression.

- pmath_t pmath_option_value (pmath_t fn, pmath_t name, pmath_t extra)

  Retrieve a option value of a given function.

11.11.1 Detailed Description

The Expression class.

Because pmath_expr_t is derived from pmath_t, you can use expressions wherever a pmath_t is accepted. E.g. you calculate an expression's hash value with pmath_hash().

The pmath_type_t of strings is PMATH_TYPE_EXPRESSION.

Expressions are arranged as array of objects. At index 0 is allways the head (function name). All arguments are at indices 1 to the length of the expression (including).

At the pMath language level, any exprssion can be entered in the form 'f(a,b,c,...)' or 'a.f(b,c,...)'. For expressions with only one argument, 'a.f' can be used instead of 'a.f()'.

11.11.2 Friends And Related Function Documentation

11.11.2.1 pmath_t pmath_evaluate_expression (pmath_expr_t *expr*, pmath_bool_t *apply_rules*) `[related]`

Partly evaluate an expression.

Parameters:

expr A pMath expression. It will be freed. Do not use it afterwards.

apply_rules Whether to apply custom rules the the whole expression or not.

Returns:

A new object produced by pMath's and user defined evaluation rules.

This function prepares an expression for evaluation (evaluates its items, ...).

Unlike pmath_evaluate(), it does not evaluate its result. In fact, pmath_evaluate() is basically a loop that calls pmath_evaluate_expression() until the result does not change any more.

The documentation for this class was generated from the following files:

- pmath-core/expressions.h
- pmath-util/emit-and-gather.h
- pmath-util/evaluation.h
- pmath-util/helpers.h

11.12 pmath_float_t Class Reference

The Floating Point Number class.

Inherits pmath_number_t.

Public Member Functions

- pmath_number_t pmath_float_new_str (const char ∗str, int base, pmath_precision_control_t precision_control, double base_precision_accuracy)

  Create a floating point number from a string.

11.12.1 Detailed Description

The Floating Point Number class.

Because pmath_float_t is derived from pmath_number_t, you can use pMath integers wherever a pmath_number_t is accepted.

The pmath_type_t of floats is `PMATH_TYPE_FLOAT`.

There are two hidden implementations of floating point numbers in pMath. One operates on `double` values. The other uses MPFR for multiple precision numbers and provides automatic precision tracking.

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

11.13 pmath_hashtable_t Class Reference

The Hashtable class.

Public Member Functions

- pmath_hashtable_t pmath_ht_create (const pmath_ht_class_t ∗klass, unsigned int minsize)

  Create a new hashtable.

- pmath_hashtable_t pmath_ht_copy (pmath_hashtable_t ht, pmath_ht_entry_copy_t entry_copy)

  Copy a given hashtable.

- void pmath_ht_destroy (pmath_hashtable_t ht)

  Destroy a given hashtable.

- void pmath_ht_clear (pmath_hashtable_t ht)

  Clear a given hashtable.

- unsigned int pmath_ht_capacity (pmath_hashtable_t ht)

  Get the capacity of a given hashtable.

- unsigned int pmath_ht_count (pmath_hashtable_t ht)

Get the number of valid entries in a given hashtable.

- void ∗ pmath_ht_entry (pmath_hashtable_t ht, unsigned int i)

  Get any entry of a given hashtable.

- void ∗ pmath_ht_search (pmath_hashtable_t ht, void ∗key)

  Search for an entry in a given hashtable.

- void ∗ pmath_ht_insert (pmath_hashtable_t ht, void ∗entry)

  Insert an entry into a given hashtable.

- void ∗ pmath_ht_remove (pmath_hashtable_t ht, void ∗key)

  Remove an entry from a given hashtable.

11.13.1 Detailed Description

The Hashtable class.

The documentation for this class was generated from the following file:

- pmath-util/hashtables.h

11.14 pmath_ht_class_t Class Reference

A hashtable interface.

Data Fields

- pmath_callback_t entry_destructor
- pmath_ht_entry_hash_func_t entry_hash
- pmath_ht_entry_equal_func_t entry_keys_equal
- pmath_ht_key_hash_func_t key_hash
- pmath_ht_entry_equals_key_func_t entry_equals_key

11.14.1 Detailed Description

A hashtable interface.

11.14.2 Field Documentation

11.14.2.1 pmath_callback_t entry_destructor

11.14.2.2 pmath_ht_entry_equals_key_func_t entry_equals_key

11.14.2.3 pmath_ht_entry_hash_func_t entry_hash

11.14.2.4 pmath_ht_entry_equal_func_t entry_keys_equal

11.14.2.5 pmath_ht_key_hash_func_t key_hash

The documentation for this class was generated from the following file:

- pmath-util/hashtables.h

11.15 pmath_integer_t Class Reference

The Integer class.

Inherits pmath_rational_t.

Public Member Functions

- pmath_integer_t pmath_integer_new_slong (signed long int si)

  Create an integer object from a signed long.

- pmath_integer_t pmath_integer_new_ulong (unsigned long int ui)

  Create an integer object from an unsigned long.

- pmath_integer_new_si32(si)

  Create an integer object from an int32_t.

- pmath_integer_t pmath_integer_new_ui32 (uint32_t ui)

  Create an integer object from an uint32_t.

- pmath_integer_t pmath_integer_new_si64 (int64_t si)

  Create an integer object from an int64_t.

- pmath_integer_t pmath_integer_new_ui64 (uint64_t ui)

  Create an integer object from an uint64_t.

- pmath_integer_new_siptr(si)

  Create an integer object from an intptr_t.

- pmath_integer_new_uiptr(ui)

  Create an integer object from an uintptr_t.

- pmath_integer_t pmath_integer_new_data (size_t count, int order, int size, int endian, size_t nails, const void ∗data)

  Create an integer object from a data buffer.

- pmath_integer_t pmath_integer_new_str (const char ∗str, int base)

  Create an integer object from a C String.

- pmath_integer_fits_si32(integer)

  Check whether a pMath integer is in range -2^31 .. 2^31-1.

- pmath_bool_t pmath_integer_fits_ui32 (pmath_integer_t integer)

  Check whether a pMath integer is in range 0 .. 2^32-1.

- pmath_bool_t pmath_integer_fits_si64 (pmath_integer_t integer)

  Check whether a pMath integer is in range -2^63 .. 2^63-1.

- pmath_bool_t pmath_integer_fits_ui64 (pmath_integer_t integer)

  Check whether a pMath integer is in range 0 .. 2^64-1.

- pmath_integer_fits_siptr(integer)

  Check whether a pMath integer fits into an intptr_t.

- pmath_integer_fits_uiptr(integer)

  Check whether a pMath integer fits into an uintptr_t.

- int32_t pmath_integer_get_si32 (pmath_integer_t integer)

  Convert a pMath integer to a signed long int.

- uint32_t pmath_integer_get_ui32 (pmath_integer_t integer)

  Convert a pMath integer to a unsigned long int.

- int64_t pmath_integer_get_si64 (pmath_integer_t integer)

  Convert a pMath integer to an int64_t.

- uint64_t pmath_integer_get_ui64 (pmath_integer_t integer)

  Convert a pMath integer to a uint64_t.

- pmath_integer_get_siptr

  Convert a pMath integer to a intptr_t.

- pmath_integer_get_uiptr

  Convert a pMath integer to a uintptr_t.

11.15.1 Detailed Description

The Integer class.

Because pmath_integer_t is derived from pmath_rational_t, you can use pMath integers wherever a pmath_rational_t is accepted.

The pmath_type_t of integers is `PMATH_TYPE_INTEGER`.

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

11.16 pmath_messages_t Class Reference

A message queue for interthread communication.

Inherits pmath_custom_t.

Related Functions

(Note that these are not member functions.)

- pmath_bool_t pmath_is_message_queue (pmath_t obj)

  Test if an object is a message queue.

- pmath_messages_t pmath_thread_get_queue (void)

  Get the current thread's message queue.

- void pmath_thread_sleep (void)

  Send the current thread to sleep.

- void pmath_thread_sleep_timeout (double abs_timeout)

  Send the current thread to sleep.

- void pmath_thread_wakeup (pmath_messages_t mq)

  Wake up another thread.

- void pmath_thread_send (pmath_messages_t mq, pmath_t msg)

  Asynchronously send a message to another thread.

- pmath_t pmath_thread_send_wait (pmath_messages_t mq, pmath_t msg, double timeout_seconds, void(∗idle_function)(void ∗), void ∗idle_data)

  Send a message to another thread and wait for the answer.

- void pmath_thread_send_delayed (pmath_messages_t mq, pmath_t msg, double seconds)

  Asynchronously send a message to a thread sometime in the future.

11.16.1 Detailed Description

A message queue for interthread communication.

The documentation for this class was generated from the following file:

- pmath-util/concurrency/threadmsg.h

11.17 pmath_number_t Class Reference

The abstract Number class.

Inherits pmath_t.

Inherited by pmath_float_t, and pmath_rational_t.

Public Member Functions

- double pmath_number_get_d (pmath_number_t number)

  Convert a pMath number to a double.

- int pmath_number_sign (pmath_number_t num)

  Get a number's sign.

- pmath_number_t pmath_number_neg (pmath_number_t num)

  Get a number's negative.

11.17.1 Detailed Description

The abstract Number class.

Because pmath_integer_t is derived from pmath_number_t, you can use pMath integers wherever a pmath_number_t is accepted.

See also:

Objects - the Base of pMath

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

11.18 pmath_quotient_t Class Reference

The Quotient class.

Inherits pmath_rational_t.

11.18.1 Detailed Description

The Quotient class.

Because pmath_quotient_t is derived from pmath_rational_t, you can use pMath integers wherever a pmath_rational_t is accepted.

The pmath_type_t of quotients is `PMATH_TYPE_QUOTIENT`.

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

11.19 pmath_rational_t Class Reference

The abstract Rational Number class.

Inherits pmath_number_t.

Inherited by pmath_integer_t, and pmath_quotient_t.

Public Member Functions

- pmath_rational_t pmath_rational_new (pmath_integer_t numerator, pmath_integer_t denominator)

  Create a rational number.

- pmath_integer_t pmath_rational_numerator (pmath_rational_t rational)

  Get the numerator of a rational number.

- pmath_integer_t pmath_rational_denominator (pmath_rational_t rational)

  Get the denominator of a rational number.

11.19.1 Detailed Description

The abstract Rational Number class.

Because pmath_rational_t is derived from pmath_number_t, you can use pMath integers wherever a pmath_number_t is accepted.

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

11.20 pmath_span_array_t Class Reference

Internal flat representaion of spans.

Public Member Functions

- void pmath_span_array_free (pmath_span_array_t *spans)

  Destroy a span-array and all its spans.

- int pmath_span_array_length (pmath_span_array_t *spans)

  Get a span-array's length.

- pmath_bool_t pmath_span_array_is_token_end (pmath_span_array_t *spans, int pos)

  Test the token-end-flag at an index.

- pmath_bool_t pmath_span_array_is_operand_start (pmath_span_array_t *spans, int pos)

  Test the operator-start-flag at an index.

- pmath_span_t * pmath_span_at (pmath_span_array_t *spans, int pos)

  Get a span starting at an index.

11.20.1 Detailed Description

Internal flat representaion of spans.

The box-form of the expression `aa*bbb+cc^-dd*ee` is `{{"aa", "*", "bbb"}, "+", {{"cc", "^", {"-", "dd"}}, "*", "ee"}}`

But those lists are not practicable for front-ends, so pMath provides a flat representation called *span-array*. It is an array of spans and flags (see pmath_span_at(), pmath_span_array_is_token_end(), pmath_span_array_is_operand_start()).

The above code would be scanned to the following span array (the text itself is not stored):

```
                aa*bbb+cc^-dd*ee
operand start:  x  x   x  xx  x
token end:       xx  xx xxx xx x
       /                  \_/
spans <         \____/ \____/
       \               \_______/
        \       \______________/
index:          0    5    10   15
```

So at index 0 is a span which ends with index 15. It has a next span (a shorter span that starts at the same position) which ends with index 5.

Another span is at index 7. It ends with index 15 and has a next span which ends with index 12.

The last span is at index 10 and also ends with index 12.

No two spans may cross-overlap so there is a definite hierachy. You create such a span-array with pmath_spans_from_string() and destroy it with pmath_span_array_free(). Conversion to and from boxes expressions can be done via pmath_boxes_from_spans() and pmath_spans_from_boxes() respectively.

The documentation for this class was generated from the following file:

- pmath-language/scanner.h

11.21 pmath_span_t Class Reference

Represents a span in a span-array.

Public Member Functions

- pmath_span_t ∗ pmath_span_next (pmath_span_t ∗span)

  Get the next-shorter span starting at the same position.

- int pmath_span_end (pmath_span_t ∗span)

  Get end of a span.

11.21.1 Detailed Description

Represents a span in a span-array.

The documentation for this class was generated from the following file:

- pmath-language/scanner.h

11.22 pmath_stack_t Class Reference

The type of pMath's threadsafe stacks.

11.22.1 Detailed Description

The type of pMath's threadsafe stacks.

You can create it with pmath_stack_new() and must destroy it with pmath_stack_free().

The documentation for this class was generated from the following file:

- pmath-util/stacks.h

11.23 pmath_string_t Class Reference

The string class.

Inherits pmath_t.

Public Member Functions

- pmath_string_t pmath_string_new (int capacity)

  Create an empty pMath String.

- pmath_string_t pmath_string_insert_latin1 (pmath_string_t str, int inspos, const char *ins, int inslen)

  Insert an Latin-1 encoded buffer into a pMath String.

- pmath_string_t pmath_string_from_utf8 (const char *str, int len)

  Convert an UTF-8 encoded buffer to a pMath String.

- char * pmath_string_to_utf8 (pmath_string_t str, int *result_len)

  Convert a pMath string to a zero-terminated UTF-8 string.

- pmath_string_t pmath_string_from_native (const char *str, int len)

  Convert a string buffer in the current console character encoding to a pMath String.

- char * pmath_string_to_native (pmath_string_t str, int *result_len)

  Convert a pMath string to a string in the current console character encoding.

- PMATH_C_STRING(cstr)

  Short form to convert a C String to a pMath String.

- pmath_string_t pmath_string_insert_codepage (pmath_string_t str, int inspos, const char *ins, int inslen, const uint16_t *cp)

  Insert a byte string into a pMath string using a translation array.

- pmath_string_t pmath_string_insert_ucs2 (pmath_string_t str, int inspos, const uint16_t *ins, int inslen)

  Insert a UCS-2 buffer into a pMath String.

- pmath_string_t pmath_string_insert (pmath_string_t str, int inspos, pmath_string_t ins)

  Insert one pMath String into another pMath String.

- pmath_string_t pmath_string_concat (pmath_string_t prefix, pmath_string_t postfix)

  Concatenate two pMath Strings.

- pmath_string_t pmath_string_part (pmath_string_t string, int start, int length)

Extract a substring of a pMath String.

- const uint16_t ∗ pmath_string_buffer (pmath_string_t ∗string)

  Get a string's buffer for reading.

- int pmath_string_length (pmath_string_t string)

  Get a string's length.

- pmath_bool_t pmath_string_equals_latin1 (pmath_string_t string, const char ∗latin1)

  Compare a pMath string with a C string.

Related Functions

(Note that these are not member functions.)

- pmath_t pmath_string_expand_boxes (pmath_string_t s)

  Expand a string that contains boxes to a list of Strings and Boxes.

- pmath_t pmath_parse_string (pmath_string_t code)

  Parse a string to an expression.

- pmath_t pmath_parse_string_args (const char ∗code, const char ∗format,...)

  Parse a string with additional arguments to an expression.

11.23.1 Detailed Description

The string class.

Because pmath_string_t is derived from pmath_t, you can use strings wherever a pmath_t is accepted. E.g. you compare two strings with pmath_compare() or pmath_equals().

The pmath_type_t of strings is PMATH_TYPE_STRING.

See also:

Objects - the Base of pMath

The documentation for this class was generated from the following files:

- pmath-core/strings.h
- pmath-language/scanner.h

11.24 pmath_symbol_t Class Reference

The Symbol class.

Inherits pmath_t.

Public Member Functions

- pmath_symbol_t pmath_symbol_get (pmath_string_t name, pmath_bool_t create)

  Get a symbol by its fully qualified name.

- pmath_symbol_t pmath_symbol_create_temporary (pmath_string_t name, pmath_bool_t unique)

  Create a new temporary symbol.

- pmath_symbol_t pmath_symbol_find (pmath_string_t name, pmath_bool_t create)

  Find a symbol in the current namespace search path.

- pmath_string_t pmath_symbol_name (pmath_symbol_t symbol)

  Get a symbol's name.

- pmath_symbol_attributes_t pmath_symbol_get_attributes (pmath_symbol_t symbol)

  Get a symbol's attributes.

- void pmath_symbol_set_attributes (pmath_symbol_t symbol, pmath_symbol_attributes_t attr)

  Set a symbol's attributes.

- pmath_t pmath_symbol_get_value (pmath_symbol_t symbol)

  Get a symbol's value.

- void pmath_symbol_set_value (pmath_symbol_t symbol, pmath_t value)

  Set a symbol's value.

- void pmath_symbol_synchronized (pmath_symbol_t symbol, pmath_callback_t callback, void ∗data)

  Execute a function synchronized to a symbol.

- void pmath_symbol_update (pmath_symbol_t symbol)

  Update a symbol manually.

- void pmath_symbol_remove (pmath_symbol_t symbol)

  Remove a symbol completely from the system.

- pmath_symbol_t pmath_symbol_iter_next (pmath_symbol_t old)

  Iterate through the global symbol table.

Related Functions

(Note that these are not member functions.)

- void pmath_collect_temporary_symbols (void)

  Collect unused symbols with the Temporary attribute.

11.24.1 Detailed Description

The Symbol class.

The pMath language knows symbols as ‘named entities‘ that can hold any pMath object as a value and have some attributes. Those symbols are objects themselves. A symbol name consists of alphanumerical characters (a-z,A-Z,0-9) and apostrophes to seperate namespaces. All standard symbols are in the "System" namespace (e.g. System‘print). All user defined symbols go to "Global". Every other module has its own namespace.

Because pmath_symbol_t is derived from pmath_t, you can use strings wherever a pmath_t is accepted. E.g. you compare two symbols with pmath_compare() or pmath_equals().

The pmath_type_t of symbols is PMATH_TYPE_SYMBOL.

See also:

Objects - the Base of pMath

11.24.2 Friends And Related Function Documentation

11.24.2.1 void pmath_collect_temporary_symbols (void) `[related]`

Collect unused symbols with the Temporary attribute.

This function is called periodically by the garbage collector.

See also:

pmath_symbol_attributes_t

The documentation for this class was generated from the following files:

- pmath-core/symbols.h
- pmath-util/concurrency/threadpool.h

11.25 pmath_t Class Reference

The basic type of all pMath objects.

Inherited by pmath_custom_t, pmath_expr_t, pmath_number_t, pmath_string_t, and pmath_symbol_t.

Public Member Functions

- pmath_bool_t pmath_equals (pmath_t objA, pmath_t objB)

  Compares two objects for identity.

- pmath_t pmath_ref (pmath_t obj)

  Increments the reference counter of an object and returns it.

- void pmath_unref (pmath_t obj)

  Decrements the reference counter of an object and frees its memory if the reference counter becomes 0.

- unsigned int pmath_hash (pmath_t obj)

  Calculates an object's hash value.

- int pmath_compare (pmath_t objA, pmath_t objB)

  Compares two objects syntactically.

- void pmath_write (pmath_t obj, pmath_write_options_t options, void(∗write)(void ∗user, const uint16_t ∗data, int len), void ∗user)

  Write an object to a stream.

- void pmath_write_ex (struct pmath_write_ex_t ∗info, pmath_t obj)

  Advanced function to write an object to a stream.

- pmath_bool_t pmath_is_evaluated (pmath_t obj)

  Test whether an object is already evaluated.

- void pmath_write_with_pagewidth (pmath_t obj, pmath_write_options_t options, void(∗write)(void ∗user, const uint16_t ∗data, int len), void ∗user, int page_width, int indentation_width)

  Write an object to a stream with a maximum line width.

Data Fields

- uint64_t as_bits
- double as_double
- struct {
  } s

Related Functions

(Note that these are not member functions.)

- pmath_t pmath_evaluate (pmath_t obj)

  Evaluate an object.

- pmath_t pmath_build_value_v (const char ∗format, va_list args)

  Generate a List of objects with a format string.

- pmath_t pmath_build_value (const char ∗format,...)

  Generate a List of objects with a format string.

11.25.1 Detailed Description

The basic type of all pMath objects.

Use pmath_is_XXX() to determine whether an object is of a specific type. Generally, you must free unused objects with pmath_unref(), but if pmath_is_pointer() gives FALSE, then calling pmath_ref() and pmath_unref() is not neccessary.

Machine precision floating point values (aka double) and certain special values are stored directly in the pmath_t object. Long strings, expressions, other values are stored as a pointer. The technique to pack all this in only 8 bytes is called NaN-boxing. See `http://blog.mozilla.com/rob-sayre/2010/08/02/mozillas-new-javascript-value-repr`

11.25.2 Member Function Documentation

11.25.2.1 pmath_bool_t pmath_equals (pmath_t *objA*, pmath_t *objB*)

Compares two objects for identity.

Parameters:

objA The first object.

objB The second one.

Returns:

TRUE iff both objects are identical.

‘identity‘ means, that X != Y is possible, even if X and Y evaluate to the same value.

If objA and objB are symbols, the result is identical to testing objA == objB.

Note:

pmath_equals(A, B) might return FALSE although pmath_compare(A, B) == 0 e.g. for an integer A and a floating point value B.

11.25.2.2 pmath_t pmath_ref (pmath_t *obj*)

Increments the reference counter of an object and returns it.

Parameters:

obj The object to be referenced.

Returns:

The referenced object. You must free the result with pmath_unref().

11.25.2.3 void pmath_unref (pmath_t *obj*)

Decrements the reference counter of an object and frees its memory if the reference counter becomes 0.

Parameters:

obj The object to be destroyed.

11.25.3 Friends And Related Function Documentation

11.25.3.1 pmath_t pmath_evaluate (pmath_t *obj*) `[related]`

Evaluate an object.

Parameters:

obj Any pMath object. It will be freed. Do not use it afterwards.

Returns:

A new object produced by pMath's and user defined evaluation rules.

11.25.4 Field Documentation

11.25.4.1 uint64_t as_bits

11.25.4.2 double as_double

11.25.4.3 struct { ... } s

The documentation for this class was generated from the following files:

- pmath-core/objects.h
- pmath-core/objects-inline.h
- pmath-util/evaluation.h
- pmath-util/helpers.h
- pmath-util/line-writer.h

11.26 pmath_text_file_api_t Class Reference

Access functions for text files.

Data Fields

- size_t struct_size

  The structure size. Allways initialize this with `sizeof(pmath_binary_file_api_t).`

- pmath_files_status_t(∗ status_function)(void ∗extra)

  A function to get the current file status This can be PMATH_NULL if read_function is also PMATH_NULL.

- pmath_string_t(∗ readln_function)(void ∗extra)

  An optional callback function for reading one line.

- pmath_bool_t(∗ write_function)(void ∗extra, const uint16_t ∗str, int len)

  An optional callback function for writing one line.

- void(∗ flush_function)(void ∗extra)

  An optional callback function for flushing an output buffer.

11.26.1 Detailed Description

Access functions for text files.

See also:

pmath_file_create_text

11.26.2 Field Documentation

11.26.2.1 void(∗ flush_function)(void ∗extra)

An optional callback function for flushing an output buffer.

11.26.2.2 pmath_string_t(∗ readln_function)(void ∗extra)

An optional callback function for reading one line.

11.26.2.3 pmath_files_status_t(∗ status_function)(void ∗extra)

A function to get the current file status This can be PMATH_NULL if read_function is also PMATH_NULL.

11.26.2.4 size_t struct_size

The structure size. Allways initialize this with `sizeof(pmath_binary_file_api_t)`.

11.26.2.5 pmath_bool_t(∗ write_function)(void ∗extra, const uint16_t ∗str, int len)

An optional callback function for writing one line.

The documentation for this class was generated from the following file:

- pmath-util/files.h

11.27 pmath_thread_t Class Reference

The Representation of a thread.

Related Functions

(Note that these are not member functions.)

- pmath_bool_t pmath_thread_queue_is_blocked_by (pmath_messages_t waiter_mq, pmath_messages_t waitee_mq)

Queries whether a thread is blocked by another thread.

- void pmath_thread_run_with_interrupt_notifier (pmath_callback_t callback, pmath_callback_t notify, void ∗callback_closure, void ∗notify_closure)

  Execute a function with an interrupt notifier installed.

- pmath_messages_t pmath_thread_fork_daemon (pmath_callback_t callback, pmath_callback_t kill, void ∗data)

  Create a new deamon thread.

- pmath_bool_t pmath_thread_fork_unmanaged (pmath_bool_t(∗init)(void ∗), void(∗callback)(void ∗), void ∗data)

  Create a new system thread.

- pmath_thread_t pmath_thread_get_current (void)

  Get the current pMath thread.

- pmath_thread_t pmath_thread_get_parent (pmath_thread_t thread)

  Get a thread's direct parent.

- pmath_bool_t pmath_thread_is_parent (pmath_thread_t parent, pmath_thread_t child)

  Queries whether a thread is one of the parents of another.

- pmath_bool_t pmath_thread_aborting (pmath_thread_t thread)

  Queries whether pMath was requested to abort the evaluation of a specific thread or its parents.

11.27.1 Detailed Description

The Representation of a thread.

Every operating system thread that runs pMath functions has its own pmath_thread_t after it successfuly initialized with pmath_init().

Todo

Implement pmath_run_parallel(number_of_parallel_threads, callback).

11.27.2 Friends And Related Function Documentation

11.27.2.1 pmath_messages_t pmath_thread_fork_daemon (pmath_callback_t *callback*, pmath_callback_t *kill*, void ∗ *data*) [related]

Create a new deamon thread.

Parameters:

callback The thread function.

kill An optional function to inform the thread that it will be killed.

data A pointer to be passed to callback() and kill()

Returns:

A reference to the new thread's message queue or PMATH_NULL on error. You have to destroy the result.

pMath will automatically kill any daemon thread when there are no other threads remaining (normaly, when pmath_done() is called from main()). Killing deamons works as follows:

```
for each deamon thread t:
    call t->kill() if the method exists
    throw PMATH_ABORT_EXCEPTION in t

for each deamon thread t:
    wait for t to finish (to return from t->callback)
```

pmath_init() will be called automatically before callback() and pmath_done() after callback returns. So the pMath thread (pmath_thread_get_current()) will be already initialized and you must not call these two functions in the *callback* routine.

You can use the *kill* function to set your own abort-please-flags if nessecary.

11.27.2.2 pmath_bool_t pmath_thread_fork_unmanaged (pmath_bool_t(∗)(void ∗) *init*, void(∗)(void ∗) *callback*, void ∗ *data*) [related]

Create a new system thread.

Parameters:

init An optional function to be called in the new thread before pmath_thread_fork_unmanaged() returns.

callback The thread function.

data A pointer to be passed to callback()

Returns:

TRUE if a new thead was created and init() returned TRUE there.

This function is just a wrapper around operating system functions. It can be used without initializing the pMath library.

The documentation for this class was generated from the following files:

- pmath-util/concurrency/threads.h
- pmath-util/concurrency/threadmsg.h
- pmath-util/concurrency/threadpool.h

11.28 pmath_threadlock_t Class Reference

A reentrant lock for threads.

Related Functions

(Note that these are not member functions.)

- void pmath_thread_call_locked (pmath_threadlock_t ∗threadlock_ptr, pmath_callback_t callback, void ∗data)

  Execute a function synchronized with a threadlock.

11.28.1 Detailed Description

A reentrant lock for threads.

A thread lock is like a thread lock, but it does not block child threads of the currently holding thread.

The documentation for this class was generated from the following file:

- pmath-util/concurrency/threadlocks.h

11.29 pmath_write_ex_t Struct Reference

Command structure for pmath_write_ex(). This should be inistialized with `memset(&ex, 0, sizeof(ex)); ex.size = sizeof(ex); ...` .

Data Fields

- size_t size

  must be initialized with sizeof(struct pmath_write_ex_t), for version control

- pmath_write_options_t options
- void(∗ write)(void ∗user, const uint16_t ∗data, int len)

  mandatory, write callback

- void ∗ user

  first parameter of the callbacks

- void(∗ pre_write)(void ∗user, pmath_t obj, pmath_write_options_t options) *optional, called before the pmath_t is written*

- void(∗ post_write)(void ∗user, pmath_t obj, pmath_write_options_t options)

  optional, called after the pmath_t is written

11.29.1 Detailed Description

Command structure for pmath_write_ex(). This should be inistialized with `memset(&ex, 0, sizeof(ex)); ex.size = sizeof(ex); ...` .

11.29.2 Field Documentation

11.29.2.1 pmath_write_options_t options

11.29.2.2 void(∗ post_write)(void ∗user, pmath_t obj, pmath_write_options_t options)

optional, called after the pmath_t is written

11.29.2.3 void(∗ pre_write)(void ∗user, pmath_t obj, pmath_write_options_t options)

optional, called before the pmath_t is written

11.29.2.4 size_t size

must be initialized with sizeof(struct pmath_write_ex_t), for version control

11.29.2.5 void∗ user

first parameter of the callbacks

11.29.2.6 void(∗ write)(void ∗user, const uint16_t ∗data, int len)

mandatory, write callback

The documentation for this struct was generated from the following file:

- pmath-core/objects.h

11.30 ReadableBinaryFile Class Reference

A wrapper for readable pMath binary file objects (byte data streams).

Inherits pmath::BinaryFile.

Public Member Functions

- ReadableBinaryFile ()
- ReadableBinaryFile (pmath_t file_object) throw ()
- ReadableBinaryFile (const Expr &file_object) throw ()
- ReadableBinaryFile (const ReadableBinaryFile &src) throw ()
- size_t read (void ∗buffer, size_t buffer_size, bool preserve_internal_buffer=false) throw ()

  Read some bytes from the file. See pmath_file_read().

Static Public Member Functions

- static ReadableBinaryFile create_uncompressor (ReadableBinaryFile srcfile) throw ()

  Create binary file object whose content is uncompressed from another binary file.

11.30.1 Detailed Description

A wrapper for readable pMath binary file objects (byte data streams).

11.30.2 Constructor & Destructor Documentation

11.30.2.1 ReadableBinaryFile () `[inline]`

11.30.2.2 ReadableBinaryFile (pmath_t *file_object*) throw () `[inline, explicit]`

11.30.2.3 ReadableBinaryFile (const Expr & *file_object*) throw () `[inline, explicit]`

11.30.2.4 ReadableBinaryFile (const ReadableBinaryFile & *src*) throw () `[inline]`

11.30.3 Member Function Documentation

11.30.3.1 static ReadableBinaryFile create_uncompressor (ReadableBinaryFile *srcfile*) throw () `[inline, static]`

Create binary file object whose content is uncompressed from another binary file.

11.30.3.2 size_t read (void ∗ *buffer*, size_t *buffer_size*, bool *preserve_internal_buffer* = `false`) throw () `[inline]`

Read some bytes from the file. See pmath_file_read().

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.31 ReadableTextFile Class Reference

A wrapper for pMath readable text file objects (byte data streams).

Inherits pmath::TextFile.

Public Member Functions

- ReadableTextFile ()
- ReadableTextFile (pmath_t file_object) throw ()
- ReadableTextFile (const Expr &file_object) throw ()
- ReadableTextFile (const ReadableTextFile &src) throw ()
- String readline () throw ()

  Read the next line from the file.

Static Public Member Functions

- static ReadableTextFile create_from_binary (ReadableBinaryFile binfile, const char ∗encoding) throw ()

  Create a text file from a binary file using a known character encoding.

- static ReadableTextFile create_from_binary (ReadableBinaryFile binfile)

  Create a text file from a binary file using UTF-16BE or UTF-16LE, depending on the machine architecture.

11.31.1 Detailed Description

A wrapper for pMath readable text file objects (byte data streams).

11.31.2 Constructor & Destructor Documentation

11.31.2.1 ReadableTextFile () `[inline]`

11.31.2.2 ReadableTextFile (pmath_t *file_object*) throw () `[inline, explicit]`

11.31.2.3 ReadableTextFile (const Expr & *file_object*) throw () `[inline, explicit]`

11.31.2.4 ReadableTextFile (const ReadableTextFile & *src*) throw () `[inline]`

11.31.3 Member Function Documentation

11.31.3.1 static ReadableTextFile create_from_binary (ReadableBinaryFile *binfile*) `[inline, static]`

Create a text file from a binary file using UTF-16BE or UTF-16LE, depending on the machine architecture.

11.31.3.2 static ReadableTextFile create_from_binary (ReadableBinaryFile *binfile*, const char ∗ *encoding*) throw () `[inline, static]`

Create a text file from a binary file using a known character encoding.

11.31.3.3 String readline () throw () `[inline]`

Read the next line from the file.

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.32 String Class Reference

A wrapper for pmath_string_t.

Inherits pmath::Expr.

Public Member Functions

- String () throw ()
- String (pmath_string_t _str) throw ()

  Construct form a pmath_string_t, stealing the reference.

- String (const Expr &src) throw ()
- String (const String &src) throw ()
- String (const char ∗latin1, int len=-1) throw ()

  Construct from Latin-1 encoded C string.

- String & operator= (const String &src) throw ()
- String & operator+= (const String &src) throw ()

  Append a string.

- String & operator+= (const char ∗latin1) throw ()

  Append a C string.

- String & operator+= (const uint16_t ∗ucs2) throw ()

  Append a UTF-16-string.

- String & operator+= (uint16_t ch) throw ()

  Append a single unicode character.

- String operator+ (const String &other) const throw ()

Concatenate two strings.

- String operator+ (const char ∗latin1) const throw ()
- String operator+ (const uint16_t ∗ucs2) const throw ()
- String operator+ (uint16_t ch) const throw ()

  Concatenate a String and a single unicode character.

- String part (int start, int length=-1) const throw ()

  Get string part.

- bool equals (const char ∗latin1) const throw ()

  Check for equality with a C string (Latin-1 encoded).

- bool starts_with (const String &s) const throw ()

  Check for prefix equality.

- bool starts_with (const char ∗latin1, int len=-1) const throw ()
- bool starts_with (const uint16_t ∗ucs2, int len=-1) const throw ()
- void insert (int pos, const String &other) throw ()

  Insert a substring. Changes the object itself.

- void insert (int pos, const char ∗latin1, int len=-1) throw ()
- void insert (int pos, const uint16_t ∗ucs2, int len=-1) throw ()
- void remove (int start, int length) throw ()

  Remove a substring.

- const String trim () const throw ()

  Trim leading and trailing whitespace.

- int length () const throw ()

  Get the string length.

- const uint16_t ∗ buffer () const throw ()

  Get the UCS-2/UTF-16 const string buffer. This is not zero-terminated.

- uint16_t operator[ ] (int i) const throw ()

  Get a single character or U+0000 on error.

- const pmath_string_t get_as_string () const throw ()

  Get the underlying pmath_string_t. It remains owned by this object.

Static Public Member Functions

- static String FromUcs2 (const uint16_t ∗ucs2, int len=-1) throw ()

  Construct from UCS-2/UTF-16 encoded string.

- static String FromChar (unsigned int unicode) throw ()

  Construct from a single unicode character.

- static String FromUtf8 (const char ∗utf8, int len=-1) throw ()

  Construct from UTF-8 encoded C string.

11.32.1 Detailed Description

A wrapper for pmath_string_t.

This class provides some string utility functions in addition to Expr.

11.32.2 Constructor & Destructor Documentation

11.32.2.1 String () throw () `[inline]`

11.32.2.2 String (pmath_string_t *_str*) throw () `[inline, explicit]`

Construct form a pmath_string_t, stealing the reference.

11.32.2.3 String (const Expr & *src*) throw () `[inline]`

11.32.2.4 String (const String & *src*) throw () `[inline]`

11.32.2.5 String (const char ∗ *latin1*, int *len* = `-1`) throw () `[inline]`

Construct from Latin-1 encoded C string.

11.32.3 Member Function Documentation

11.32.3.1 const uint16_t∗ buffer () const throw () `[inline]`

Get the UCS-2/UTF-16 const string buffer. This is not zero-terminated.

11.32.3.2 bool equals (const char ∗ *latin1*) const throw () `[inline]`

Check for equality with a C string (Latin-1 encoded).

11.32.3.3 static String FromChar (unsigned int *unicode*) throw () `[inline, static]`

Construct from a single unicode character.

11.32.3.4 static String FromUcs2 (const uint16_t ∗ *ucs2*, int *len* = `-1`) throw () `[inline, static]`

Construct from UCS-2/UTF-16 encoded string.

11.32.3.5 static String FromUtf8 (const char ∗ *utf8*, int *len* = `-1`) throw () `[inline, static]`

Construct from UTF-8 encoded C string.

11.32.3.6 const pmath_string_t get_as_string () const throw () `[inline]`

Get the underlying pmath_string_t. It remains owned by this object.

11.32.3.7 void insert (int *pos*, const uint16_t ∗ *ucs2*, int *len* = `-1`) throw () `[inline]`

11.32.3.8 void insert (int *pos*, const char ∗ *latin1*, int *len* = `-1`) throw () `[inline]`

11.32.3.9 void insert (int *pos*, const String & *other*) throw () `[inline]`

Insert a substring. Changes the object itself.

11.32.3.10 int length () const throw () `[inline]`

Get the string length.

11.32.3.11 String operator+ (uint16_t *ch*) const throw () `[inline]`

Concatenate a String and a single unicode character.

11.32.3.12 String operator+ (const uint16_t ∗ *ucs2*) const throw () `[inline]`

11.32.3.13 String operator+ (const char ∗ *latin1*) const throw () `[inline]`

11.32.3.14 String operator+ (const String & *other*) const throw () `[inline]`

Concatenate two strings.

11.32.3.15 String& operator+= (uint16_t *ch*) throw () `[inline]`

Append a single unicode character.

11.32.3.16 String& operator+= (const uint16_t ∗ *ucs2*) throw () `[inline]`

Append a UTF-16-string.

11.32.3.17 String& operator+= (const char ∗ *latin1*) throw () `[inline]`

Append a C string.

11.32.3.18 String& operator+= (const String & *src*) throw () `[inline]`

Append a string.

11.32.3.19 String& operator= (const String & *src*) throw () `[inline]`

11.32.3.20 uint16_t operator[ ] (int *i*) const throw () `[inline]`

Get a single character or U+0000 on error.

11.32.3.21 String part (int *start*, int *length* = `-1`) const throw () `[inline]`

Get string part.

11.32.3.22 void remove (int *start*, int *length*) throw () `[inline]`

Remove a substring.

11.32.3.23 bool starts_with (const uint16_t ∗ *ucs2*, int *len* = `-1`) const throw () `[inline]`

11.32.3.24 bool starts_with (const char ∗ *latin1*, int *len* = `-1`) const throw () `[inline]`

11.32.3.25 bool starts_with (const String & *s*) const throw () `[inline]`

Check for prefix equality.

11.32.3.26 const String trim () const throw () `[inline]`

Trim leading and trailing whitespace.

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.33 TextFile Class Reference

A wrapper for pMath text file objects (byte data streams).

Inherits pmath::File.

Inherited by ReadableTextFile, and WriteableTextFile.

Public Member Functions

- TextFile ()
- TextFile (pmath_t file_object) throw ()
- TextFile (const Expr &file_object) throw ()
- TextFile (const TextFile &src) throw ()
- void set_buffer (String buffer)

  Set a file's internal text buffer.

Static Public Member Functions

- static TextFile create_from_binary (BinaryFile binfile, const char ∗encoding) throw ()

  Create a text file from a binary file using a known character encoding.

- static TextFile create_from_binary (BinaryFile binfile)

  Create a text file from a binary file using UTF-16BE or UTF-16LE, depending on the machine architecture.

11.33.1 Detailed Description

A wrapper for pMath text file objects (byte data streams).

11.33.2 Constructor & Destructor Documentation

11.33.2.1 TextFile () `[inline]`

11.33.2.2 TextFile (pmath_t *file_object*) throw () `[inline, explicit]`

11.33.2.3 TextFile (const Expr & *file_object*) throw () `[inline, explicit]`

11.33.2.4 TextFile (const TextFile & *src*) throw () `[inline]`

11.33.3 Member Function Documentation

11.33.3.1 static TextFile create_from_binary (BinaryFile *binfile*) `[inline, static]`

Create a text file from a binary file using UTF-16BE or UTF-16LE, depending on the machine architecture.

11.33.3.2 static TextFile create_from_binary (BinaryFile *binfile*, const char ∗ *encoding*) throw () `[inline, static]`

Create a text file from a binary file using a known character encoding.

11.33.3.3 void set_buffer (String *buffer*) `[inline]`

Set a file's internal text buffer.

Parameters:

buffer The new line buffer. It should not contain any newline character!

See also:

pmath_file_set_textbuffer().

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.34 TextUserStream Class Reference

Abstract base class for C++ callbacks used as pMath text files.

Inherits pmath::UserStream.

Public Member Functions

- virtual pmath_files_status_t status ()=0

  Called by pMath to check for end-of-file and other errors.

- virtual void flush ()

  Called by pMath to flush data to disk.

- virtual String readline ()=0

  Called by pMath to read a line of text, excluding any newline characters.

- virtual bool write (const uint16_t ∗str, int len)=0

  Called by pMath to write text.

Protected Member Functions

- TextFile convert_to_file (bool readable, bool writeable)
- ReadableTextFile convert_to_file_readonly ()
- WriteableTextFile convert_to_file_writeonly ()

11.34.1 Detailed Description

Abstract base class for C++ callbacks used as pMath text files.

11.34.2 Member Function Documentation

11.34.2.1 TextFile convert_to_file (bool *readable*, bool *writeable*) `[inline, protected]`

11.34.2.2 ReadableTextFile convert_to_file_readonly () `[inline, protected]`

11.34.2.3 WriteableTextFile convert_to_file_writeonly () `[inline, protected]`

11.34.2.4 virtual void flush () `[inline, virtual]`

Called by pMath to flush data to disk.

11.34.2.5 virtual String readline () `[pure virtual]`

Called by pMath to read a line of text, excluding any newline characters.

11.34.2.6 virtual pmath_files_status_t status () `[pure virtual]`

Called by pMath to check for end-of-file and other errors.

11.34.2.7 virtual bool write (const uint16_t * *str*, int *len*) `[pure virtual]`

Called by pMath to write text.

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.35 UserStream Class Reference

Abstract base class for C++ callbacks used as pMath files.

Inherited by BinaryUserStream, and TextUserStream.

Public Member Functions

- virtual ~UserStream ()

Static Public Member Functions

- template<class U >
  static bool file_wraps (File file)

  Test whether a pMath file wraps a user stream.

Protected Member Functions

- virtual void dereference ()

  Called by pMath when the object is no longer needed.

Static Protected Member Functions

- template<class U , typename A >
  static bool manipulate (File file, void(U::∗callback)(const A &), const A &arg)

  Call a method on the user stream behind a pMath file.

- template<class U >
  static bool manipulate (BinaryFile file, void(U::∗callback)())

  Call a method on the user stream behind a pMath file.

- static void destructor_function (void ∗extra)

  Called by pMath.

11.35.1 Detailed Description

Abstract base class for C++ callbacks used as pMath files.

The object destructor must be thread-safe (e.g. by not using any global data), because it is typically called from another thread than where the object was created. If synchronization is needed, it can be done in the dereference() callback method.

All other callback methods are synchronized: pMath ensures that no callback is entered twice at the same time.

11.35.2 Constructor & Destructor Documentation

11.35.2.1 virtual ∼UserStream () `[inline, virtual]`

11.35.3 Member Function Documentation

11.35.3.1 virtual void dereference () `[inline, protected, virtual]`

Called by pMath when the object is no longer needed.

This method destroys the object by default.

11.35.3.2 static void destructor_function (void $*$ *extra*) `[inline, static, protected]`

Called by pMath.

11.35.3.3 static bool file_wraps (File *file*) `[inline, static]`

Test whether a pMath file wraps a user stream.

Parameters:

file The pMath file.

Returns:

true iff *file* wraps UserStream subclass U object

11.35.3.4 static bool manipulate (BinaryFile *file*, void(U::$*$)() *callback*) `[inline, static, protected]`

Call a method on the user stream behind a pMath file.

Parameters:

file A pMath file that was created from a user stream class U.

callback A member method of the user stream function Uto be called by pMath.

Returns:

Whether the callback was called. That is, whether the file is actually a user stream of class U.

11.35.3.5 static bool manipulate (File *file*, void(U::∗)(const A &) *callback*, const A & *arg*) [inline, static, protected]

Call a method on the user stream behind a pMath file.

Parameters:

file A pMath file that was created from a user stream class U.

callback A member method of the user stream function Uto be called by pMath.

arg An argument to the callback.

Returns:

Whether the callback was called. That is, whether the file is actually a user stream of class U.

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.36 WriteableBinaryFile Class Reference

A wrapper for writeable pMath binary file objects (byte data streams).

Inherits pmath::BinaryFile.

Public Member Functions

- WriteableBinaryFile ()
- WriteableBinaryFile (pmath_t file_object) throw ()
- WriteableBinaryFile (const Expr &file_object) throw ()
- WriteableBinaryFile (const WriteableBinaryFile &src) throw ()
- size_t write (const void ∗buffer, size_t buffer_size) throw ()

  Write some bytes to the file. See pmath_file_write().

Static Public Member Functions

- static WriteableBinaryFile create_compressor (WriteableBinaryFile dstfile) throw ()

  Create binary file object whose content is compressed into another binary file.

11.36.1 Detailed Description

A wrapper for writeable pMath binary file objects (byte data streams).

11.36.2 Constructor & Destructor Documentation

11.36.2.1 WriteableBinaryFile () `[inline]`

11.36.2.2 WriteableBinaryFile (pmath_t *file_object*) throw () `[inline, explicit]`

11.36.2.3 WriteableBinaryFile (const Expr & *file_object*) throw () `[inline, explicit]`

11.36.2.4 WriteableBinaryFile (const WriteableBinaryFile & *src*) throw () `[inline]`

11.36.3 Member Function Documentation

11.36.3.1 static WriteableBinaryFile create_compressor (WriteableBinaryFile *dstfile*) throw () `[inline, static]`

Create binary file object whose content is compressed into another binary file.

11.36.3.2 size_t write (const void $*$ *buffer*, size_t *buffer_size*) throw () `[inline]`

Write some bytes to the file. See pmath_file_write().

The documentation for this class was generated from the following file:

- pmath-cpp.h

11.37 WriteableTextFile Class Reference

A wrapper for pMath writeable text file objects (byte data streams).

Inherits pmath::TextFile.

Public Member Functions

- WriteableTextFile () throw ()
- WriteableTextFile (pmath_t file_object) throw ()
- WriteableTextFile (const Expr &file_object) throw ()
- WriteableTextFile (const WriteableTextFile &src) throw ()
- void write (String str) throw ()

  Write some text to the file.

Static Public Member Functions

- static WriteableTextFile create_from_binary (WriteableBinaryFile binfile, const char ∗encoding) throw ()

  Create a text file from a binary file using a known character encoding.

- static WriteableTextFile create_from_binary (WriteableBinaryFile binfile) throw ()

  Create a text file from a binary file using UTF-16BE or UTF-16LE, depending on the machine architecture.

11.37.1 Detailed Description

A wrapper for pMath writeable text file objects (byte data streams).

11.37.2 Constructor & Destructor Documentation

11.37.2.1 WriteableTextFile () throw () `[inline]`

11.37.2.2 WriteableTextFile (pmath_t *file_object*) throw () `[inline, explicit]`

11.37.2.3 WriteableTextFile (const Expr & *file_object*) throw () `[inline, explicit]`

11.37.2.4 WriteableTextFile (const WriteableTextFile & *src*) throw () `[inline]`

11.37.3 Member Function Documentation

11.37.3.1 static WriteableTextFile create_from_binary (WriteableBinaryFile *binfile*) throw () `[inline, static]`

Create a text file from a binary file using UTF-16BE or UTF-16LE, depending on the machine architecture.

11.37.3.2 static WriteableTextFile create_from_binary (WriteableBinaryFile *binfile*, const char ∗ *encoding*) throw () `[inline, static]`

Create a text file from a binary file using a known character encoding.

11.37.3.3 void write (String *str*) throw () `[inline]`

Write some text to the file.

The documentation for this class was generated from the following file:

- pmath-cpp.h

12 File Documentation

12.1 pmath-config.h File Reference

Defines

- #define PMATH_CONCAT_(a, b) a##b
- #define PMATH_CONCAT(a, b) PMATH_CONCAT_(a, b)
- #define PMATH_STATIC_ASSERT(e) typedef char PMATH_CONCAT(pmath_static_assert_line_, __LINE__)[(e)?1:-1]
- #define PMATH_NEED_GNUC(maj, min) (0)
- #define PMATH_ATTRIBUTE_PURE
- #define PMATH_ATTRIBUTE_USE_RESULT
- #define PMATH_ATTRIBUTE_NONNULL(...)
- #define PMATH_DEPRECATED
- #define PMATH_LIKELY(cond) (cond)
- #define PMATH_UNLIKELY(cond) (cond)
- #define PMATH_UNUSED
- #define PMATH_FORCE_INLINE static __inline
- #define PMATH_INLINE __inline
- #define PMATH_EXTERN_C

- #define PMATH_MODULE PMATH_EXTERN_C __attribute__((__visibility__("default")))
- #define PMATH_API PMATH_EXTERN_C __attribute__((__visibility__("default")))
- #define PMATH_BYTE_ORDER 1

12.1.1 Define Documentation

12.1.1.1 #define PMATH_API PMATH_EXTERN_C __attribute__((__visibility__("default")))

12.1.1.2 #define PMATH_ATTRIBUTE_NONNULL(...)

12.1.1.3 #define PMATH_ATTRIBUTE_PURE

12.1.1.4 #define PMATH_ATTRIBUTE_USE_RESULT

12.1.1.5 #define PMATH_BYTE_ORDER 1

12.1.1.6 #define PMATH_CONCAT(a, b) PMATH_CONCAT_(a, b)

12.1.1.7 #define PMATH_CONCAT_(a, b) a##b

12.1.1.8 #define PMATH_DEPRECATED

12.1.1.9 #define PMATH_EXTERN_C

12.1.1.10 #define PMATH_FORCE_INLINE static __inline

12.1.1.11 #define PMATH_INLINE __inline

12.1.1.12 #define PMATH_LIKELY(cond) (cond)

12.1.1.13 #define PMATH_MODULE PMATH_EXTERN_C __attribute__((__visibility__("default")))

12.1.1.14 #define PMATH_NEED_GNUC(maj, min) (0)

12.1.1.15 #define PMATH_STATIC_ASSERT(e) typedef char PMATH_CONCAT(pmath_static_assert_line_, __LINE__)[(e)?1:-1]

12.1.1.16 #define PMATH_UNLIKELY(cond) (cond)

12.1.1.17 #define PMATH_UNUSED

12.2 pmath-core/custom.h File Reference

Typedefs

- typedef pmath_t pmath_custom_t

12.3 pmath-core/expressions.h File Reference

Typedefs

- typedef pmath_t pmath_expr_t

12.4 pmath-core/numbers.h File Reference

Defines

- #define PMATH_MACHINE_PRECISION 0
- #define PMATH_AUTO_PRECISION 1
- #define pmath_integer_new_si32(si) PMATH_FROM_INT32(si)

  Create an integer object from an int32_t.
- #define pmath_integer_new_siptr(si)

  Create an integer object from an intptr_t.
- #define pmath_integer_new_uiptr(ui)

  Create an integer object from an uintptr_t.
- #define pmath_integer_fits_si32(integer) pmath_is_int32(integer)

  Check whether a pMath integer is in range -2^31 .. 2^31-1.
- #define pmath_integer_fits_siptr(integer)

  Check whether a pMath integer fits into an intptr_t.
- #define pmath_integer_fits_uiptr(integer)

  Check whether a pMath integer fits into an uintptr_t.
- #define pmath_integer_get_siptr

  Convert a pMath integer to a intptr_t.
- #define pmath_integer_get_uiptr

  Convert a pMath integer to a uintptr_t.

Typedefs

- typedef pmath_t pmath_number_t
- typedef pmath_number_t pmath_rational_t
- typedef pmath_rational_t pmath_integer_t
- typedef pmath_rational_t pmath_mpint_t
- typedef pmath_rational_t pmath_quotient_t
- typedef pmath_number_t pmath_float_t
- typedef pmath_float_t pmath_mpfloat_t

Enumerations

- enum pmath_precision_control_t {
  PMATH_PREC_CTRL_AUTO = 0, PMATH_PREC_CTRL_MACHINE_PREC = 1,
  PMATH_PREC_CTRL_GIVEN_PREC = 2, PMATH_PREC_CTRL_GIVEN_ACC = 3 }

12.5 pmath-core/objects-inline.h File Reference

Defines

- #define pmath_is_double(obj) (((obj).s.tag & PMATH_TAGMASK_NONDOUBLE) != PMATH_TAGMASK_NONDOUBLE)
- #define pmath_is_pointer(obj) (((obj).s.tag & PMATH_TAGMASK_POINTER) == PMATH_TAGMASK_POINTER)
- #define pmath_is_magic(obj) ((obj).s.tag == PMATH_TAG_MAGIC)
- #define pmath_is_int32(obj) ((obj).s.tag == PMATH_TAG_INT32)
- #define pmath_is_str0(obj) ((obj).s.tag == PMATH_TAG_STR0)
- #define pmath_is_str1(obj) ((obj).s.tag == PMATH_TAG_STR1)
- #define pmath_is_str2(obj) ((obj).s.tag == PMATH_TAG_STR2)
- #define pmath_is_ministr(obj) (pmath_is_str0(obj) || pmath_is_str1(obj) || pmath_is_str2(obj))
- #define PMATH_AS_TAG(obj) ((obj).s.tag)
- #define PMATH_AS_INT32(obj) ((obj).s.u.as_int32)
- #define pmath_same(objA, objB) ((objA).as_bits == (objB).as_bits)
- #define pmath_is_null(obj) (pmath_same(obj, PMATH_NULL))
- #define pmath_is_mpint(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_MP_INT))
- #define pmath_is_mpfloat(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_MP_FLOAT))
- #define pmath_is_custom(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_CUSTOM))
- #define pmath_is_expr(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_EXPRESSION))
- #define pmath_is_float(obj) (pmath_is_double(obj) || pmath_is_mpfloat(obj))
- #define pmath_is_integer(obj) (pmath_is_int32(obj) || pmath_is_mpint(obj))
- #define pmath_is_number(obj) (pmath_is_float(obj) || pmath_is_rational(obj))
- #define pmath_is_quotient(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_QUOTIENT))
- #define pmath_is_rational(obj) (pmath_is_integer(obj) || pmath_is_quotient(obj))
- #define pmath_is_bigstr(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_BIGSTRING))
- #define pmath_is_string(obj) (pmath_is_ministr(obj) || pmath_is_bigstr(obj))
- #define pmath_is_symbol(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_SYMBOL))

Functions

- struct _pmath_t ∗ PMATH_AS_PTR (pmath_t obj)
- double PMATH_AS_DOUBLE (pmath_t obj)
- pmath_bool_t pmath_is_pointer_of (pmath_t obj, pmath_type_t type)
- pmath_bool_t pmath_is_evaluatable (pmath_t obj)
- pmath_t PMATH_FROM_DOUBLE (double d)
- intptr_t pmath_refcount (pmath_t obj)

12.5.1 Define Documentation

12.5.1.1 #define PMATH_AS_INT32(obj) ((obj).s.u.as_int32)

12.5.1.2 #define PMATH_AS_TAG(obj) ((obj).s.tag)

12.5.1.3 #define pmath_is_bigstr(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_BIGSTRING))

12.5.1.4 #define pmath_is_custom(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_CUSTOM))

12.5.1.5 #define pmath_is_double(obj) (((obj).s.tag & PMATH_TAGMASK_NONDOUBLE) != PMATH_TAGMASK_NONDOUBLE)

12.5.1.6 #define pmath_is_expr(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_EXPRESSION))

12.5.1.7 #define pmath_is_float(obj) (pmath_is_double(obj) || pmath_is_mpfloat(obj))

12.5.1.8 #define pmath_is_int32(obj) ((obj).s.tag == PMATH_TAG_INT32)

12.5.1.9 #define pmath_is_integer(obj) (pmath_is_int32(obj) || pmath_is_mpint(obj))

12.5.1.10 #define pmath_is_magic(obj) ((obj).s.tag == PMATH_TAG_MAGIC)

12.5.1.11 #define pmath_is_ministr(obj) (pmath_is_str0(obj) || pmath_is_str1(obj) || pmath_is_str2(obj))

12.5.1.12 #define pmath_is_mpfloat(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_MP_FLOAT))

12.5.1.13 #define pmath_is_mpint(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_MP_INT))

12.5.1.14 #define pmath_is_null(obj) (pmath_same(obj, PMATH_NULL))

12.5.1.15 #define pmath_is_number(obj) (pmath_is_float(obj) || pmath_is_rational(obj))

12.5.1.16 #define pmath_is_pointer(obj) (((obj).s.tag & PMATH_TAGMASK_-POINTER) == PMATH_TAGMASK_POINTER)

12.5.1.17 #define pmath_is_quotient(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_QUOTIENT))

12.5.1.18 #define pmath_is_rational(obj) (pmath_is_integer(obj) || pmath_is_quotient(obj))

12.5.1.19 #define pmath_is_str0(obj) ((obj).s.tag == PMATH_TAG_STR0)

12.5.1.20 #define pmath_is_str1(obj) ((obj).s.tag == PMATH_TAG_STR1)

12.5.1.21 #define pmath_is_str2(obj) ((obj).s.tag == PMATH_TAG_STR2)

12.5.1.22 #define pmath_is_string(obj) (pmath_is_ministr(obj) || pmath_is_bigstr(obj))

12.5.1.23 #define pmath_is_symbol(obj) (pmath_is_pointer_of(obj, PMATH_TYPE_SYMBOL))

12.5.1.24 #define pmath_same(objA, objB) ((objA).as_bits == (objB).as_bits)

12.5.2 Function Documentation

12.5.2.1 double PMATH_AS_DOUBLE (pmath_t *obj*)

12.5.2.2 struct _pmath_t∗ PMATH_AS_PTR (pmath_t *obj*) `[read]`

12.5.2.3 pmath_t PMATH_FROM_DOUBLE (double *d*)

12.5.2.4 pmath_bool_t pmath_is_evaluatable (pmath_t *obj*)

12.5.2.5 pmath_bool_t pmath_is_pointer_of (pmath_t *obj*, pmath_type_t *type*)

12.5.2.6 intptr_t pmath_refcount (pmath_t *obj*)

12.6 pmath-core/objects.h File Reference

Data Structures

- class pmath_t

  The basic type of all pMath objects.

- struct pmath_write_ex_t

  Command structure for pmath_write_ex(). This should be inistialized with `memset(&ex, 0, sizeof(ex)); ex.size = sizeof(ex); ...` .

Defines

- #define PMATH_TAGMASK_BITCOUNT 12
- #define PMATH_TAGMASK_NONDOUBLE 0x7FF00000U
- #define PMATH_TAGMASK_POINTER 0xFFF00000U
- #define PMATH_TAG_INVALID (PMATH_TAGMASK_NONDOUBLE | 0xFFFFF)
- #define PMATH_TAG_MAGIC (PMATH_TAGMASK_NONDOUBLE | 0x10000)
- #define PMATH_TAG_INT32 (PMATH_TAGMASK_NONDOUBLE | 0x20000)
- #define PMATH_TAG_STR0 (PMATH_TAGMASK_NONDOUBLE | 0x30000)

- #define PMATH_TAG_STR1 (PMATH_TAGMASK_NONDOUBLE | 0x40000)
- #define PMATH_TAG_STR2 (PMATH_TAGMASK_NONDOUBLE | 0x50000)
- #define PMATH_THREAD_KEY_PARSESYMBOLS PMATH_FROM_TAG(PMATH_TAG_MAGIC, 252)
- #define PMATH_THREAD_KEY_PARSERARGUMENTS PMATH_FROM_TAG(PMATH_TAG_MAGIC, 253)
- #define PMATH_ABORT_EXCEPTION PMATH_FROM_TAG(PMATH_TAG_MAGIC, 254)
- #define PMATH_STATIC_UNDEFINED { (((uint64_t)PMATH_TAG_MAGIC) << 32) | 255 }
- #define PMATH_STATIC_NULL { ((uint64_t)PMATH_TAGMASK_POINTER) << 32 }

Typedefs

- typedef int pmath_type_t

  The type or class of a pMath object.

- typedef int pmath_write_options_t

  Options for pmath_write().

- typedef void(∗ pmath_proc_t)(pmath_t)

  A simple procedure operating on an object.

- typedef void(∗ pmath_param_proc_t)(void ∗, pmath_t)

  A parameterized procedure operating on an object.

- typedef pmath_t(∗ pmath_func_t)(pmath_t)

  A simple function operating on an object and returning one.

- typedef unsigned int(∗ pmath_hash_func_t)(pmath_t)

  A hash function for an object.

- typedef pmath_bool_t(∗ pmath_equal_func_t)(pmath_t, pmath_t)

  A comparision function for two objects.

- typedef int(∗ pmath_compare_func_t)(pmath_t, pmath_t)

  A comparision function to determine the order of two objects.

Enumerations

- enum {

  PMATH_TYPE_SHIFT_MP_FLOAT = 0, PMATH_TYPE_SHIFT_MP_INT, PMATH_TYPE_SHIFT_QUOTIENT, PMATH_TYPE_SHIFT_BIGSTRING,

PMATH_TYPE_SHIFT_SYMBOL, PMATH_TYPE_SHIFT_EXPRESSION_GENERAL,

PMATH_TYPE_SHIFT_EXPRESSION_GENERAL_PART, PMATH_TYPE_SHIFT_RESERVED_1,

PMATH_TYPE_SHIFT_CUSTOM, PMATH_TYPE_SHIFT_COUNT }

- enum {

  PMATH_TYPE_MP_INT = 1 << PMATH_TYPE_SHIFT_MP_INT, PMATH_TYPE_QUOTIENT = 1 << PMATH_TYPE_SHIFT_QUOTIENT,

  PMATH_TYPE_MP_FLOAT = 1 << PMATH_TYPE_SHIFT_MP_FLOAT, PMATH_TYPE_BIGSTRING = 1 << PMATH_TYPE_SHIFT_BIGSTRING,

  PMATH_TYPE_SYMBOL = 1 << PMATH_TYPE_SHIFT_SYMBOL, PMATH_TYPE_EXPRESSION_GENERAL = 1 << PMATH_TYPE_SHIFT_EXPRESSION_GENERAL,

  PMATH_TYPE_EXPRESSION_GENERAL_PART = 1 << PMATH_TYPE_SHIFT_EXPRESSION_GENERAL_PART, PMATH_TYPE_EXPRESSION = PMATH_TYPE_EXPRESSION_GENERAL | PMATH_TYPE_EXPRESSION_GENERAL_PART,

  PMATH_TYPE_CUSTOM = 1 << PMATH_TYPE_SHIFT_CUSTOM }
- enum {

  PMATH_WRITE_OPTIONS_FULLEXPR = 1 << 0, PMATH_WRITE_OPTIONS_FULLSTR = 1 << 1,

  PMATH_WRITE_OPTIONS_FULLNAME = 1 << 2, PMATH_WRITE_OPTIONS_INPUTEXPR = 1 << 3 }

Functions

- pmath_t PMATH_FROM_TAG (uint32_t tag, int32_t value)
- pmath_t PMATH_FROM_INT32 (int32_t i)
- pmath_t PMATH_FROM_PTR (void ∗p)
- size_t pmath_object_bytecount (pmath_t obj)

  Get the byte count of an object.

Variables

- static PMATH_UNUSED const pmath_t PMATH_UNDEFINED

  Magic value to indicate unset variable values/...

- static PMATH_UNUSED const pmath_t PMATH_NULL

  The NULL pointer. /\/ in pMath.

12.7 pmath-core/strings.h File Reference

Data Structures

- struct pmath_cstr_writer_info_t

  Additional information for pmath_utf8_writer() or pmath_native_writer().

Defines

- #define PMATH_C_STRING(cstr) pmath_string_insert_latin1(PMATH_NULL, 0, (cstr), -1)

  Short form to convert a C String to a pMath String.

Typedefs

- typedef pmath_t pmath_string_t

Functions

- void pmath_utf8_writer (void ∗user, const uint16_t ∗data, int len)

  A write *function for pmath_write() that converts to utf8.*

- void pmath_native_writer (void ∗user, const uint16_t ∗data, int len)

  A write *function for pmath_write() that converts to the current console encosing.*

12.8 pmath-core/symbols.h File Reference

Typedefs

- typedef pmath_t pmath_symbol_t
- typedef int pmath_symbol_attributes_t

  The (bitset) type of symbol attributes.

Enumerations

- enum {

  PMATH_SYMBOL_ATTRIBUTE_PROTECTED = 1 << 0, PMATH_SYMBOL_ATTRIBUTE_HOLDFIRST = 1 << 1,

  PMATH_SYMBOL_ATTRIBUTE_HOLDREST = 1 << 2, PMATH_SYMBOL_ATTRIBUTE_HOLDALL = PMATH_SYMBOL_ATTRIBUTE_HOLDFIRST | PMATH_SYMBOL_ATTRIBUTE_HOLDREST, PMATH_SYMBOL_ATTRIBUTE_SYMMETRIC = 1 << 3, PMATH_SYMBOL_ATTRIBUTE_ASSOCIATIVE = 1 << 4,

PMATH_SYMBOL_ATTRIBUTE_NHOLDFIRST = 1 << 5, PMATH_SYMBOL_ATTRIBUTE_NHOLDREST = 1 << 6,

PMATH_SYMBOL_ATTRIBUTE_NHOLDALL = PMATH_SYMBOL_ATTRIBUTE_NHOLDFIRST | PMATH_SYMBOL_ATTRIBUTE_NHOLDREST, PMATH_SYMBOL_ATTRIBUTE_TEMPORARY = 1 << 7,

PMATH_SYMBOL_ATTRIBUTE_LISTABLE = 1 << 8, PMATH_SYMBOL_ATTRIBUTE_DEEPHOLDALL = 1 << 9,

PMATH_SYMBOL_ATTRIBUTE_HOLDALLCOMPLETE = 1 << 10, PMATH_SYMBOL_ATTRIBUTE_ONEIDENTITY = 1 << 11,

PMATH_SYMBOL_ATTRIBUTE_THREADLOCAL = 1 << 12, PMATH_SYMBOL_ATTRIBUTE_NUMERICFUNCTION = 1 << 13,

PMATH_SYMBOL_ATTRIBUTE_READPROTECTED = 1 << 14, PMATH_SYMBOL_ATTRIBUTE_SEQUENCEHOLD = 1 << 15,

PMATH_SYMBOL_ATTRIBUTE_REMOVED = 1 << 16, PMATH_SYMBOL_ATTRIBUTE_DEFINITEFUNCTION = 1 << 17 }

12.9 pmath-cpp.h File Reference

Data Structures

- class Expr

  A wrapper for pmath_t and drived types.

- class String

  A wrapper for pmath_string_t.

- class Gather

  Utility class for emitting and gathering expressions/building lists.

- class File

  A wrapper for pMath file objects (data streams).

- class BinaryFile

  A wrapper for pMath binary file objects (byte data streams).

- class ReadableBinaryFile

  A wrapper for readable pMath binary file objects (byte data streams).

- class WriteableBinaryFile

  A wrapper for writeable pMath binary file objects (byte data streams).

- class TextFile

A wrapper for pMath text file objects (byte data streams).

- class ReadableTextFile

  A wrapper for pMath readable text file objects (byte data streams).

- class WriteableTextFile

  A wrapper for pMath writeable text file objects (byte data streams).

- class UserStream

  Abstract base class for C++ callbacks used as pMath files.

- class BinaryUserStream

  Abstract base class for C++ callbacks used as pMath binary files.

- class TextUserStream

  Abstract base class for C++ callbacks used as pMath text files.

Namespaces

- namespace pmath

  Provides the C++ binding.

Functions

- Expr Number (double d)
- Expr Complex (const Expr &re, const Expr &im)
- Expr Imaginary (const Expr &im)
- Expr Rational (const Expr &num, const Expr &den)
- Expr Ref (pmath_t o)
- Expr Symbol (pmath_symbol_t h)
- Expr SymbolPi ()
- Expr MakeList (size_t len)
- Expr Call (const Expr &h)
- Expr Call (const Expr &h, const Expr &x1)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7)

- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8)
- Expr Call (const Expr &h, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8, const Expr &x9)
- Expr List ()
- Expr List (const Expr &x1)
- Expr List (const Expr &x1, const Expr &x2)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8)
- Expr List (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4, const Expr &x5, const Expr &x6, const Expr &x7, const Expr &x8, const Expr &x9)
- Expr Rule (const Expr &l, const Expr &r)
- Expr RuleDelayed (const Expr &l, const Expr &r)
- Expr Power (const Expr &x, const Expr &y)
- Expr Sqrt (const Expr &x)
- Expr Inv (const Expr &x)
- Expr Exp (const Expr &x)
- Expr Log (const Expr &x)
- Expr Log (const Expr &b, const Expr &x)
- Expr Sin (const Expr &x)
- Expr Cos (const Expr &x)
- Expr Tan (const Expr &x)
- Expr ArcSin (const Expr &x)
- Expr ArcCos (const Expr &x)
- Expr ArcTan (const Expr &x)
- Expr Times (const Expr &x1, const Expr &x2)
- Expr Times (const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Times (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr Divide (const Expr &x, const Expr &y)
- Expr Plus (const Expr &x1, const Expr &x2)
- Expr Plus (const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Plus (const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)
- Expr Minus (const Expr &x)
- Expr Minus (const Expr &x, const Expr &y)
- Expr Abs (const Expr &x)
- Expr Arg (const Expr &x)

- Expr Sign (const Expr &x)
- Expr Re (const Expr &x)
- Expr Im (const Expr &x)
- Expr Ceiling (const Expr &x)
- Expr Ceiling (const Expr &x, const Expr &a)
- Expr Floor (const Expr &x)
- Expr Floor (const Expr &x, const Expr &a)
- Expr Round (const Expr &x)
- Expr Round (const Expr &x, const Expr &a)
- Expr Quotient (const Expr &m, const Expr &n)
- Expr Quotient (const Expr &m, const Expr &n, const Expr &d)
- Expr Mod (const Expr &m, const Expr &n)
- Expr Mod (const Expr &m, const Expr &n, const Expr &d)
- Expr Evaluate (const Expr &x)
- Expr ParseArgs (const char ∗code, const Expr &arglist)
- Expr Parse (const String &code)
- Expr Parse (const char ∗code)
- Expr Parse (const char ∗code, const Expr &x1)
- Expr Parse (const char ∗code, const Expr &x1, const Expr &x2)
- Expr Parse (const char ∗code, const Expr &x1, const Expr &x2, const Expr &x3)
- Expr Parse (const char ∗code, const Expr &x1, const Expr &x2, const Expr &x3, const Expr &x4)

12.10 pmath-language/charnames.h File Reference

Functions

- uint32_t pmath_char_from_name (const char ∗name)

  Get a named character.

- const char ∗ pmath_char_to_name (uint32_t unichar)

  Get a character's name.

- const uint16_t ∗ pmath_char_parse (const uint16_t ∗str, int maxlen, uint32_t ∗result)

  Parse an escaped character to a unicode codepoint.

12.11 pmath-language/scanner.h File Reference

Defines

- #define PMATH_RUN(code)

  Execute some pMath code.

- #define PMATH_RUN_ARGS(code, format,...)

  Execute some pMath code with arguments.

Typedefs

- typedef struct _pmath_span_array_t pmath_span_array_t
- typedef struct _pmath_span_t pmath_span_t

Functions

- pmath_span_array_t ∗ pmath_spans_from_string (pmath_string_t ∗code, pmath_string_t(∗line_reader)(void ∗), pmath_bool_t(∗subsuperscriptbox_at_index)(int, void ∗), pmath_string_t(∗underoverscriptbox_at_index)(int, void ∗), void(∗error)(pmath_string_t, int, void ∗, pmath_bool_t), void ∗data)

  Parses pMath code to a span array.

- pmath_t pmath_boxes_from_spans (pmath_span_array_t ∗spans, pmath_string_t string, pmath_bool_t parseable, pmath_t(∗box_at_index)(int, void ∗), void ∗data)

  Convert a span-array with the according code to boxed form.

- pmath_span_array_t ∗ pmath_spans_from_boxes (pmath_t boxes, pmath_string_t ∗result_string, void(∗make_box)(int, pmath_t, void ∗), void ∗data)

  Convert boxed form back to span-array and code.

12.12 pmath-language/tokens.h File Reference

Defines

- #define PMATH_CHAR_INVISIBLECALL 0x2061

  The Function application character.

- #define PMATH_CHAR_VECTOR 0x21C0

  The arrow above names to indicate a vector.

- #define PMATH_CHAR_RULE 0x2192

  The " → " operator.

- #define PMATH_CHAR_RULEDELAYED 0x29F4

  The ":>" operator.

- #define PMATH_CHAR_ASSIGN 0x2254

  The ":=" operator.

- #define PMATH_CHAR_ASSIGNDELAYED 0x2A74

  The "::=" operator.

- #define PMATH_CHAR_INTEGRAL_D 0x2146

The integral "d".

- #define PMATH_CHAR_PIECEWISE 0xF361

  The left curly bracket for cases.

- #define PMATH_CHAR_ALIASDELIMITER 0xF764

  The character inserted by Richmath with ESCAPE or CAPSLOCK.

- #define PMATH_CHAR_ALIASINDICATOR 0xF768

  A character that looks like PMATH_CHAR_ALIASDELIMITER but has no effect.

- #define PMATH_CHAR_LEFT_BOX 0xFFF9

  Start of box code inside a string.

- #define PMATH_CHAR_RIGHT_BOX 0xFFFB

  End of box code inside a string.

- #define PMATH_CHAR_BOX 0xFDD0

  Represents a box.

- #define PMATH_CHAR_PLACEHOLDER 0xFFFD

  The Placeholder character. In richmath, type CAPSLOCK pl CAPSLOCK to insert it.

Enumerations

- enum pmath_token_t {

  PMATH_TOK_NONE, PMATH_TOK_SPACE,

  PMATH_TOK_DIGIT, PMATH_TOK_STRING,

  PMATH_TOK_NAME, PMATH_TOK_NAME2,

  PMATH_TOK_BINARY_LEFT, PMATH_TOK_BINARY_RIGHT,

  PMATH_TOK_BINARY_LEFT_AUTOARG, PMATH_TOK_BINARY_LEFT_OR_PREFIX,

  PMATH_TOK_NARY, PMATH_TOK_NARY_AUTOARG,

  PMATH_TOK_NARY_OR_PREFIX, PMATH_TOK_POSTFIX_OR_PREFIX,

  PMATH_TOK_PREFIX, PMATH_TOK_POSTFIX,

  PMATH_TOK_CALL, PMATH_TOK_LEFTCALL,

  PMATH_TOK_LEFT, PMATH_TOK_RIGHT,

  PMATH_TOK_PRETEXT, PMATH_TOK_ASSIGNTAG,

  PMATH_TOK_PLUSPLUS, PMATH_TOK_COLON,

  PMATH_TOK_TILDES, PMATH_TOK_SLOT,

  PMATH_TOK_QUESTION, PMATH_TOK_INTEGRAL,

  PMATH_TOK_COMMENTEND }

Token classes known in the pMath language.

- enum {
  PMATH_PREC_ANY = 0, PMATH_PREC_SEQ = 10,
  PMATH_PREC_EVAL = 20, PMATH_PREC_ASS = 30,
  PMATH_PREC_MODY = 40, PMATH_PREC_LAZY = 50,
  PMATH_PREC_FUNC = 60, PMATH_PREC_REPL = 80,
  PMATH_PREC_RULE = 90, PMATH_PREC_MAP = 100,
  PMATH_PREC_STR = 110, PMATH_PREC_COND = 120,
  PMATH_PREC_ALT = 130, PMATH_PREC_OR = 150,
  PMATH_PREC_XOR = 155, PMATH_PREC_AND = 160,
  PMATH_PREC_ARROW = 170, PMATH_PREC_REL = 180,
  PMATH_PREC_UNION = 190, PMATH_PREC_ISECT = 200,
  PMATH_PREC_RANGE = 210, PMATH_PREC_ADD = 220,
  PMATH_PREC_CIRCADD = 230, PMATH_PREC_PLUMI = 240,
  PMATH_PREC_CIRCMUL = 250, PMATH_PREC_MUL = 260,
  PMATH_PREC_DIV = 270, PMATH_PREC_MIDDOT = 280,
  PMATH_PREC_CROSS = 290, PMATH_PREC_MUL2 = 300,
  PMATH_PREC_POW = 310, PMATH_PREC_FAC = 320,
  PMATH_PREC_APL = 330, PMATH_PREC_REPEAT = 340,
  PMATH_PREC_TEST = 350, PMATH_PREC_INC = 360,
  PMATH_PREC_CALL = 400, PMATH_PREC_DIFF = 410,
  PMATH_PREC_PRIM = 1000 }

Functions

- pmath_token_t pmath_token_analyse (const uint16_t ∗str, int len, int ∗prec)

  Analyse a token.

- int pmath_token_prefix_precedence (const uint16_t ∗str, int len, int defprec)

  Give the prefix oprator precedence for a token.

- static PMATH_INLINE pmath_bool_t pmath_token_maybe_first (pmath_token_t tok)

  Test whether a token may be the first token in a subexpression.

- static PMATH_INLINE pmath_bool_t pmath_token_maybe_rest (pmath_token_t tok)

  Test whether a token need not be the first token in a subexpression.

- static PMATH_INLINE pmath_bool_t pmath_char_is_left (uint16_t ch)

Test if a unicode character is a left bracket.

- static PMATH_INLINE uint16_t pmath_right_fence (uint16_t left)

  Get the corresponding right bracket to a given left bracket or 0.

- static PMATH_INLINE pmath_bool_t pmath_char_is_right (uint16_t ch)

  Test if a unicode character is a right bracket.

- static PMATH_INLINE pmath_bool_t pmath_char_is_name (uint16_t ch)

  Test if a unicode character can be the start of an identifier/name.

- static PMATH_INLINE pmath_bool_t pmath_char_is_integral (uint16_t ch)

  Test if a unicode character is an integral.

- static PMATH_INLINE pmath_bool_t pmath_token_maybe_bigop (pmath_token_t tok)

  Test if a token may be a big operator.

- static PMATH_INLINE pmath_bool_t pmath_char_maybe_bigop (uint16_t ch)

  Test if a unicode character may be a big operation, e.g. Union, Sum.

- static PMATH_INLINE pmath_bool_t pmath_char_is_digit (uint16_t ch)

  Test if a unicode character is a digit '0' - '9'.

- static PMATH_INLINE pmath_bool_t pmath_char_is_36digit (uint16_t ch)

  Test if a unicode character is a base-36 digit '0' - '9', 'a' - 'z', 'A' - 'Z'.

- static PMATH_INLINE pmath_bool_t pmath_char_is_basedigit (int base, uint16_t ch)

  Test if in a given base, a unicode character is a digit.

- static PMATH_INLINE pmath_bool_t pmath_char_is_hexdigit (uint16_t ch)

  Test if a unicode character is a hexadecimal digit.

12.13 pmath-types.h File Reference

Defines

- #define FALSE ((pmath_bool_t)0)

  The FALSE value for pmath_bool_t.

- #define TRUE (!FALSE)

  The TRUE value for pmath_bool_t.

- #define PMATH_INVALID_PTR ((void∗)UINTPTR_MAX)

Typedefs

- typedef char pmath_bool_t

  A boolean type.

- typedef void(∗ pmath_callback_t)(void ∗)

  A general callback function.

12.14 pmath-util/approximate.h File Reference

Functions

- pmath_bool_t pmath_is_numeric (pmath_t obj)

  Test whether an expression is a numeric quantity.

- double pmath_accuracy (pmath_t obj)

  Get the accuracy (in bits) of an object.

- double pmath_precision (pmath_t obj)

  Get the precision (in bits) of an object.

- pmath_t pmath_set_accuracy (pmath_t obj, double acc)

  Set an object's accuracy in bits.

- pmath_t pmath_set_precision (pmath_t obj, double prec)

  Set an object's accuracy in bits.

- pmath_t pmath_approximate (pmath_t obj, double precision_goal, double accuracy_goal, pmath_bool_t ∗aborted)

  Approximate an object.

12.15 pmath-util/compression.h File Reference

Functions

- pmath_symbol_t pmath_file_create_compressor (pmath_t dstfile)

  Create a writeable binary file object that compresses its input.

- pmath_symbol_t pmath_file_create_uncompressor (pmath_t srcfile)

  Create a readable binary file object that uncompresses its input.

12.16 pmath-util/concurrency/atomic.h File Reference

Data Structures

- struct pmath_atomic_t

  A pointer-sized atomic variable type.

- struct pmath_atomic2_t

  A 2-pointer-sized atomic variable type.

Defines

- #define PMATH_ATOMIC_STATIC_INIT {0};
- #define PMATH_ATOMIC_FASTLOOP_COUNT (1000)
- #define pmath_atomic_loop_yield() (sched_yield())
- #define pmath_atomic_loop_nop()

12.16.1 Define Documentation

12.16.1.1 #define PMATH_ATOMIC_FASTLOOP_COUNT (1000)

12.16.1.2 #define pmath_atomic_loop_nop(void)

Value:

```
do{ \
      struct timespec tm; \
      tm.tv_sec = 0; \
      tm.tv_nsec = 2000001; \
      nanosleep(&tm, NULL); \
    }while(0)
```

12.16.1.3 #define pmath_atomic_loop_yield(void) (sched_yield())

12.16.1.4 #define PMATH_ATOMIC_STATIC_INIT {0};

12.17 pmath-util/concurrency/atomic/non-atomic.h File Reference

Defines

- #define PMATH_ATOMIC_FASTLOOP_COUNT (0)

  Loop iterations in spinlocks before yielding control.

- #define PMATH_DECLARE_ALIGNED(TYPE, NAME, ALIGNMENT) TYPE NAME

  Declares a variable with specified alignment.

Functions

- intptr_t pmath_atomic_fetch_add (pmath_atomic_t ∗atom, intptr_t delta)

  Add a value to another.

- intptr_t pmath_atomic_fetch_set (pmath_atomic_t ∗atom, intptr_t new_value)

  Exchange a value.

- intptr_t pmath_atomic_fetch_compare_and_set (pmath_atomic_t ∗atom, intptr_t old_value, intptr_t new_value)

  Exchange a value if it equals another value.

- pmath_bool_t pmath_atomic_compare_and_set (pmath_atomic_t ∗atom, intptr_t old_value, intptr_t new_value)

  Exchange a value if it equals another value.

- pmath_bool_t pmath_atomic_compare_and_set_2 (pmath_atomic2_t ∗atom, intptr_t old_value_fst, intptr_t old_value_snd, intptr_t new_value_fst, intptr_t new_value_snd)

  Exchange two values value if they equal another two values.

- pmath_bool_t pmath_atomic_have_cas2 (void)

  Check, whether the CPU supports pmath_atomic_compare_and_set_2().

- void pmath_atomic_barrier (void)

  Insert an explicit memory barrier.

- void pmath_atomic_lock (pmath_atomic_t ∗atom)

  Try to aquire a lock.

- void pmath_atomic_unlock (pmath_atomic_t ∗atom)

  Release a previously aquired lock.

- void pmath_atomic_loop_yield (void)

  Yield control to another thread (used in spinlocks).

- void pmath_atomic_loop_nop (void)

  A no-operation or short wait for use in spin locks.

12.18 pmath-util/concurrency/threadlocks.h File Reference

Typedefs

- typedef struct _pmath_threadlock_t ∗ pmath_threadlock_t

12.19 pmath-util/concurrency/threadmsg.h File Reference

Typedefs

- typedef pmath_custom_t pmath_messages_t

Functions

- double pmath_tickcount (void)

  Gives the seconds since January 1, 1970 (UTC).

12.20 pmath-util/concurrency/threadpool.h File Reference

Typedefs

- typedef struct _pmath_task_t ∗ pmath_task_t

Functions

- pmath_task_t pmath_task_ref (pmath_task_t task)
- void pmath_task_unref (pmath_task_t task)
- void ∗ pmath_task_get_data (pmath_task_t task)
- pmath_bool_t pmath_task_has_destructor (pmath_task_t task, pmath_callback_t dtor)
- pmath_task_t pmath_task_new (pmath_callback_t run, pmath_callback_t destroy, void ∗data)
- void pmath_task_wait (pmath_task_t task)

12.20.1 Typedef Documentation

12.20.1.1 typedef struct _pmath_task_t∗ pmath_task_t

Todo

document pmath-util/concurrency/threadpool.h

12.20.2 Function Documentation

12.20.2.1 void∗ pmath_task_get_data (pmath_task_t *task*)

12.20.2.2 pmath_bool_t pmath_task_has_destructor (pmath_task_t *task*, pmath_callback_t *dtor*)

12.20.2.3 pmath_task_t pmath_task_new (pmath_callback_t *run*, pmath_callback_t *destroy*, void ∗ *data*)

12.20.2.4 pmath_task_t pmath_task_ref (pmath_task_t *task*)

12.20.2.5 void pmath_task_unref (pmath_task_t *task*)

12.20.2.6 void pmath_task_wait (pmath_task_t *task*)

12.21 pmath-util/concurrency/threads.h File Reference

Typedefs

- typedef struct _pmath_thread_t ∗ pmath_thread_t

Functions

- pmath_t pmath_thread_local_save (pmath_t key, pmath_t value)

  Store a thread/thread-local value.

- pmath_t pmath_thread_local_load (pmath_t key)

Load a thread/thread-local value.

- void pmath_throw (pmath_t exception)

  Throw an exception.

- pmath_t pmath_catch (void)

  Catch any exception.

- pmath_bool_t pmath_aborting (void)

  Queries whether pMath was requested to abort the evaluation of the current thread.

- void pmath_abort_please (void)

  Requests pMath to abort the current evaluation.

- pmath_bool_t pmath_continue_after_abort (void)

  Requests pMath to stop aborting the current evaluation.

- void pmath_suspend_all_please (void)

  Suspend all other threads. This function does not realy suspend threads immediately. Any other thread, that calls pmath_aborting() (or pmath_thread_aborting()), will block until we call pmath_resume_all().

- void pmath_resume_all (void)

  Resume all other threads.

12.22 pmath-util/debug.h File Reference

Functions

- void pmath_debug_print (const char ∗fmt,...)

  Print out a simple debug message.

- void pmath_debug_print_object (const char ∗pre, pmath_t obj, const char ∗post)

  Print a pMath object to the debug log.

- void pmath_debug_print_stack (void)

  Print the current pMath stack trace to the debug log.

12.23 pmath-util/dlmalloc.h File Reference

12.24 pmath-util/emit-and-gather.h File Reference

12.25 pmath-util/evaluation.h File Reference

12.26 pmath-util/files.h File Reference

Data Structures

- class pmath_binary_file_api_t

  Access functions for binary files.

- class pmath_text_file_api_t

  Access functions for text files.

Enumerations

- enum pmath_files_status_t {
  PMATH_FILE_OK = 0, PMATH_FILE_INVALID = 1,
  PMATH_FILE_ENDOFFILE = 2, PMATH_FILE_OTHERERROR = 3,
  PMATH_FILE_RECURSIVE = 4 }

  The status of a file.

Functions

- pmath_bool_t pmath_file_test (pmath_t file, int properties)

  Check whether a file supports a set of properties.

- pmath_files_status_t pmath_file_status (pmath_t file)

  Get the current status of a readable file.

- size_t pmath_file_read (pmath_t file, void ∗buffer, size_t buffer_size, pmath_bool_t preserve_internal_buffer)

  Read some bytes from a binary file.

- pmath_string_t pmath_file_readline (pmath_t file)

  Read one line from a text file.

- void pmath_file_set_textbuffer (pmath_t file, pmath_string_t buffer)

  Set a file's internal text buffer.

- size_t pmath_file_write (pmath_t file, const void ∗buffer, size_t buffer_size)

Write some bytes to a binary file.

- pmath_bool_t pmath_file_writetext (pmath_t file, const uint16_t ∗str, int len)

  Write to a text file.

- void pmath_file_flush (pmath_t file)

  Flush the output buffer of a writeable file.

- pmath_bool_t pmath_file_write_object (pmath_t file, pmath_t obj, pmath_write_options_t options)

  Write an object to a text file.

- pmath_bool_t pmath_file_set_binbuffer (pmath_t file, size_t size)

  Set a binary file's buffer size.

- void pmath_file_manipulate (pmath_t file, void(∗type)(void ∗), void(∗callback)(void ∗, void ∗), void ∗data)

  Manipulate a file's internal representation.

- pmath_bool_t pmath_file_close (pmath_t file)

  Closes a file.

- void pmath_file_close_if_unused (pmath_t file)

  Closes a file if it is not referenced somewhere else.

- pmath_symbol_t pmath_file_create_binary (void ∗extra, void(∗extra_destructor)(void ∗), pmath_binary_file_api_t ∗api)

  Create a binary file object.

- pmath_symbol_t pmath_file_create_text (void ∗extra, void(∗extra_destructor)(void ∗), pmath_text_file_api_t ∗api)

  Create a text file object.

- pmath_symbol_t pmath_file_create_text_from_binary (pmath_t binfile, const char ∗encoding)

  Create a text file object operating on a binary file.

- pmath_symbol_t pmath_file_create_binary_buffer (size_t mincapacity)

  Create a byte-stream file object.

- size_t pmath_file_binary_buffer_size (pmath_t binfile)

  Get The number of readable bytes in a binary buffer.

- void pmath_file_binary_buffer_manipulate (pmath_t binfile, void(∗callback)(uint8_t ∗readable, uint8_t ∗∗writable, const uint8_t ∗end, void ∗closure), void ∗closure)

  Manipulate the content of a binary buffer.

12.27 pmath-util/hashtables.h File Reference

Data Structures

- class pmath_ht_class_t

  A hashtable interface.

Typedefs

- typedef struct _pmath_hashtable_t ∗ pmath_hashtable_t
- typedef void(∗ pmath_ht_entry_callback_t)(void ∗entry, void ∗data)

  A callback function for hashtable entries.

- typedef void ∗(∗ pmath_ht_entry_copy_t)(void ∗entry)

  An entry copy function.

- typedef unsigned int(∗ pmath_ht_entry_hash_func_t)(void ∗entry)

  An entry hash function.

- typedef unsigned int(∗ pmath_ht_key_hash_func_t)(void ∗key)

  A key hash function.

- typedef pmath_bool_t(∗ pmath_ht_entry_equal_func_t)(void ∗entry1, void ∗entry2)

  An entry comparision function.

- typedef pmath_bool_t(∗ pmath_ht_entry_equals_key_func_t)(void ∗entry, void ∗key)

  An entry to key comparision function.

12.28 pmath-util/helpers.h File Reference

Typedefs

- typedef pmath_bool_t(∗ pmath_stack_walker_t)(pmath_t head, void ∗closure)

  A stack walker function.

Functions

- pmath_bool_t pmath_is_expr_of (pmath_t obj, pmath_symbol_t head)

  Check if an object is an expression with a specified head.

- pmath_bool_t pmath_is_expr_of_len (pmath_t obj, pmath_symbol_t head, size_t length)

  Check if an object is an expression with a specified head and length.

- pmath_t pmath_current_head (void)

  Get the currently evaluated function.

- void pmath_walk_stack (pmath_stack_walker_t walker, void ∗closure)

  Walk up the current thread's and its parents' stack.

- pmath_t pmath_session_execute (pmath_t input, pmath_bool_t ∗aborted)

  Execute an expression and change $History and $Line appropriately.

- pmath_t pmath_session_start (void)

  Saves some global state when an interactive dialog session starts.

- void pmath_session_end (pmath_t old_state)

  Restore some global state when an interactive dialog session ends.

12.29 pmath-util/line-writer.h File Reference

12.30 pmath-util/memory.h File Reference

Functions

- void ∗ pmath_mem_alloc (size_t size)

  Allocate some amount of memory.

- void ∗ pmath_mem_realloc (void ∗p, size_t new_size)

  Change the size of a memory-chunk.

- void ∗ pmath_mem_realloc_no_failfree (void ∗p, size_t new_size)

  Change the size of a memory-chunk.

- void pmath_mem_free (void ∗p)

  Free a memory-chunk.

- void pmath_mem_usage (size_t ∗current, size_t ∗max)

  Get memory usage information.

12.31 pmath-util/messages.h File Reference

Functions

- void pmath_message (pmath_symbol_t symbol, const char ∗tag, size_t argcount,...)

  Print a message with pMath object arguments.

- void pmath_message_argxxx (size_t given, size_t min, size_t max)

  Generate a General::arg∗ message (invalid argument count).

- pmath_string_t pmath_message_find_text (pmath_t name)

  Find a message's text.

- void pmath_message_syntax_error (pmath_string_t code, int position, pmath_string_t filename, int lines_before_code)

  Print a syntax warning or error message.

12.32 pmath-util/mixed-file.h File Reference

Functions

- void pmath_file_create_mixed_buffer (const char ∗encoding, pmath_symbol_t ∗out_textfile, pmath_symbol_t ∗out_binfile)

  Creates a mixed binary/text file double ended queue.

12.33 pmath-util/serialize.h File Reference

Enumerations

- enum pmath_serialize_error_t {
  PMATH_SERIALIZE_OK = 0, PMATH_SERIALIZE_NO_MEMORY = 1,
  PMATH_SERIALIZE_BAD_OBJECT = 2, PMATH_SERIALIZE_EOF = 3,
  PMATH_SERIALIZE_BAD_BYTE = 4, PMATH_SERIALIZE_BAD_REF = 5
  }

  (De-)Serialization error codes.

Functions

- pmath_serialize_error_t pmath_serialize (pmath_t file, pmath_t object)

  Write an object to a binary file.

- pmath_t pmath_deserialize (pmath_t file, pmath_serialize_error_t ∗error)

  Write an object to a binary file.

12.33.1 Enumeration Type Documentation

12.33.1.1 enum pmath_serialize_error_t

(De-)Serialization error codes.

Enumerator:

PMATH_SERIALIZE_OK No error occured.

PMATH_SERIALIZE_NO_MEMORY

PMATH_SERIALIZE_BAD_OBJECT The object cannot be serialized (e.g. custom objects).

PMATH_SERIALIZE_EOF Unexpected end of file.

PMATH_SERIALIZE_BAD_BYTE Unexpected byte.

PMATH_SERIALIZE_BAD_REF Unknown back reference.

12.33.2 Function Documentation

12.33.2.1 pmath_t pmath_deserialize (pmath_t *file*, pmath_serialize_error_t ∗ *error*)

Write an object to a binary file.

Parameters:

file A file object. It wont be freed.

error Where to put the error code (optional).

Returns:

The deserialized object.

12.33.2.2 pmath_serialize_error_t pmath_serialize (pmath_t *file*, pmath_t *object*)

Write an object to a binary file.

Parameters:

file A file object. It wont be freed.

object A pMath object. It will be freed.

Returns:

An error code.

12.34 pmath-util/stacks.h File Reference

Typedefs

- typedef struct _pmath_stack_t ∗ pmath_stack_t

Functions

- pmath_stack_t pmath_stack_new (void)
  Create an empty stack.
- void pmath_stack_free (pmath_stack_t stack)
  Destroy a stack.
- void pmath_stack_push (pmath_stack_t stack, void ∗item)
  Push an item onto a stack.
- void ∗ pmath_stack_pop (pmath_stack_t stack)
  Pop an item from a stack.

12.35 pmath-util/strtod.h File Reference

Functions

- double pmath_strtod (const char ∗str, const char ∗∗end)
  locale-neutral strtod

12.35.1 Function Documentation

12.35.1.1 double pmath_strtod (const char ∗ *str*, const char ∗∗ *end*)

locale-neutral strtod

12.36 pmath-util/version.h File Reference

Functions

- void pmath_version_datetime (int ∗year, int ∗month, int ∗day, int ∗hour, int ∗minute, int ∗second)

  Get the date and time when pMath was compiled.

- double pmath_version_number (void)

  Get version number (major + minor/100).

- long pmath_version_number_part (int index)

  Get version number part.

12.37 pmath.h File Reference

Functions

- pmath_bool_t pmath_init (void)

  Initialize the pMath CAS library.

- void pmath_done (void)

  Free all resources used by the pMath CAS library and unload all modules.

Index