pMath

0.1

Generated by Doxygen 1.5.9

Sun Apr 18 17:46:43 2010

Contents

1 The pMath Computer Algebra System Library

Author:

Peter Frentrup

Date:

2008

Introduction

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

- The pMath library documented here, which implements the parser, interpreter, mathematical functionality and OS binding.
- The Richmath library for a graphical front-end.

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.

Links/Depencies

pMath is build on top of several open source libraries:

- GMP (`http://gmplib.org`)
- MPFR (`http://www.mpfr.org`)
- iconv (e.g. `http://www.gnu.org/software/libiconv`)
- PCRE (`http://www.pcre.org`)

2 Todo List

Class pmath_thread_t Implement pmath_run_parallel(number_of_parallel_threads, callback).

Group cpp_binding Document the Expr helper functions.

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, void *dat

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Data Structure Index

5.1 Class Hierarchy

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

6 Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

7 Module Documentation

7.1 Custom Objects

Encapsulate arbitrary data in pMath objects.

Data Structures

- class pmath_custom_t

  The Custom Object class.

Functions

- PMATH_API pmath_custom_t pmath_custom_t::pmath_custom_new (void ∗data, pmath_callback_t destructor)

  Create a custom object.

- PMATH_API void ∗ pmath_custom_t::pmath_custom_get_data (pmath_custom_t custom)

  Get a custom object's data member.

- PMATH_API pmath_bool_t pmath_custom_t::pmath_custom_has_destructor (pmath_custom_t custom, pmath_callback_t dtor)

  Get a custom object's data destructor.

7.1.1 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 NULL. But you can store custom objects in symbols (directly with 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))));
```

7.1.2 Function Documentation

7.1.2.1 PMATH_API 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 NULL on failure (in that case, *destructor(data)* is called immediately).

7.1.2.2 PMATH_API 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 NULL if *custom* is 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 onlyx 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 are stored in one symbols. 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 (Multithreading with pMath), you must also store a synchronization object (e.g. symbol or threadlock) in the *data* member und use this.

7.1.2.3 PMATH_API pmath_bool_t pmath_custom_has_destructor (pmath_custom_t *custom*, pmath_callback_t *dtor*) [inherited]

Get a custom object's data destructor.

Parameters:

custom A custom object.

dtor A callback function.

Returns:

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

7.2 Expressions

Expression objects in pMath.

Data Structures

- class pmath_expr_t

  The Expression class.

Functions

- PMATH_API pmath_expr_t pmath_expr_t::pmath_expr_new (pmath_t head, size_t length)

  Create a new expression.

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

  Create a new expression with all items given.

- PMATH_API pmath_expr_t pmath_expr_t::pmath_expr_resize (pmath_expr_t expr, size_t new_length)

  Resize an expression.

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

  Append some items to an expression.

- PMATH_API size_t pmath_expr_t::pmath_expr_length (pmath_expr_t expr)

  Get an expression's length.

- PMATH_API pmath_t pmath_expr_t::pmath_expr_get_item (pmath_expr_t expr, size_t index)

  Get an item from an expression.

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

Get multiple items from an expression.

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

  Set an item in an expression.

- PMATH_API pmath_expr_t pmath_expr_t::pmath_expr_remove_all (pmath_expr_t expr, pmath_t rem)

  Remove all occurencies of an object from an expression.

- PMATH_API pmath_expr_t pmath_expr_t::pmath_expr_sort (pmath_expr_t expr)

  Sort an expression.

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

  Flatten an expression.

7.2.1 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

7.2.2 Function Documentation

7.2.2.1 PMATH_API 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:

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

7.2.2.2 PMATH_API 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:

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().

7.2.2.3 PMATH_API 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:

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 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 NULL.

7.2.2.4 PMATH_API 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:

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 == NULL, the returns value is pmath_expr_new_extended(NULL,count,...).

7.2.2.5 PMATH_API 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).

7.2.2.6 PMATH_API 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 NULL otherwise. You must destroy it with pmath_unref().

7.2.2.7 PMATH_API 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*.

7.2.2.8 PMATH_API 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:

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.

7.2.2.9 PMATH_API 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:

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

7.2.2.10 PMATH_API 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).

7.2.2.11 PMATH_API 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.

7.3 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.

Functions

- PMATH_API double pmath_accuracy (pmath_t obj)

  Get the accuracy (in bits) of an object.

- PMATH_API double pmath_precision (pmath_t obj)

  Get the precision (in bits) of an object.

- PMATH_API pmath_t pmath_set_accuracy (pmath_t obj, double acc)

  Set an object’s accuracy in bits.

- PMATH_API pmath_t pmath_set_precision (pmath_t obj, double prec)

  Set an object's accuracy in bits.

- PMATH_API pmath_t pmath_approximate (pmath_t obj, double prec, double acc)

  Approximate an object.

- PMATH_API pmath_integer_t pmath_integer_t::pmath_integer_new_si (signed long int si)

  Create an integer object from a signed long.

- PMATH_API pmath_integer_t pmath_integer_t::pmath_integer_new_ui (unsigned long int ui)

  Create an integer object from an unsigned long.

- PMATH_API pmath_integer_t pmath_integer_t::pmath_integer_new_size (size_t size)

  Create an integer object from an size_t.

- PMATH_API pmath_integer_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_API pmath_integer_t pmath_integer_t::pmath_integer_new_str (const char ∗str, int base)

  Create an integer object from a C String.

- PMATH_API pmath_rational_t pmath_rational_t::pmath_rational_new (pmath_integer_t numerator, pmath_integer_t denominator)

  Create a rational number.

- PMATH_API pmath_integer_t pmath_rational_t::pmath_rational_numerator (pmath_rational_t rational)

  Get the numerator of a rational number.

- PMATH_API pmath_integer_t pmath_rational_t::pmath_rational_denominator (pmath_rational_-t rational)

  Get the denominator of a rational number.

- PMATH_API pmath_number_t pmath_float_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_API pmath_float_t pmath_float_t::pmath_float_new_d (double dbl)

  Create a machine-precision floating point number from a double.

- PMATH_API pmath_bool_t pmath_integer_t::pmath_integer_fits_si (pmath_integer_t integer)

  Find out whether a pMath integer fits into a signed long int.

- PMATH_API pmath_bool_t pmath_integer_t::pmath_integer_fits_ui (pmath_integer_t integer)

  Find out whether a pMath integer fits into a unsigned long int.

- PMATH_API signed long int pmath_integer_t::pmath_integer_get_si (pmath_integer_t integer)

  Convert a pMath integer to a signed long int.

- PMATH_API unsigned long int pmath_integer_t::pmath_integer_get_ui (pmath_integer_t integer)

  Convert a pMath integer to a unsigned long int.

- PMATH_API double pmath_number_t::pmath_number_get_d (pmath_number_t number)

  Convert a pMath number to a double.

- PMATH_API int pmath_number_t::pmath_number_sign (pmath_number_t num)

  Get a number's sign.

- PMATH_API pmath_number_t pmath_number_t::pmath_number_neg (pmath_number_t num)

  Get a number's negative.

7.3.1 Detailed Description

Number objects in pMath.

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

7.3.2 Function Documentation

7.3.2.1 PMATH_API 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.

7.3.2.2 PMATH_API 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.

7.3.2.3 PMATH_API 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.

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

7.3.2.4 PMATH_API 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.

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

7.3.2.5 PMATH_API pmath_t pmath_approximate (pmath_t *obj*, double *prec*, double *acc*)

Approximate an object.

Parameters:

- ***obj*** An object. It will be freed.
- ***prec*** The requested precision in bits.
- ***acc*** The requested accurarcy in bits.

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.

7.3.2.6 PMATH_API pmath_integer_t pmath_integer_new_si (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 NULL.

7.3.2.7 PMATH_API pmath_integer_t pmath_integer_new_ui (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 NULL.

7.3.2.8 PMATH_API pmath_integer_t pmath_integer_new_size (size_t *size*) `[inherited]`

Create an integer object from an size_t.

Parameters:

size A size_t value.

Returns:

A pMath integer with the specified value or NULL.

Note that on Win64, sizeof(long) == 4, but sizeof(size_t) == 8.

7.3.2.9 PMATH_API 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()

7.3.2.10 PMATH_API 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 NULL.

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

7.3.2.11 PMATH_API 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, NULL will be returned.

7.3.2.12 PMATH_API 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().

7.3.2.13 PMATH_API 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().

7.3.2.14 PMATH_API 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 precinion or accuracy. depending on the value of the above flag.

Returns:

a new pMath floating point number or 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)).`

7.3.2.15 PMATH_API pmath_float_t pmath_float_new_d (double *dbl*) `[inherited]`

Create a machine-precision floating point number from a double.

Parameters:

dbl A double value.

Returns:

A pMath floating point number with the specified value or NULL.

7.3.2.16 PMATH_API pmath_bool_t pmath_integer_fits_si (pmath_integer_t *integer*) `[inherited]`

Find out whether a pMath integer fits into a signed long int.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough for a signed long int.

7.3.2.17 PMATH_API pmath_bool_t pmath_integer_fits_ui (pmath_integer_t *integer*) `[inherited]`

Find out whether a pMath integer fits into a unsigned long int.

Parameters:

integer A pMath integer. It wont be freed.

Returns:

TRUE iff the value is small enough for a unsigned long int.

7.3.2.18 PMATH_API signed long int pmath_integer_get_si (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_si

7.3.2.19 PMATH_API unsigned long int pmath_integer_get_ui (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_ui

7.3.2.20 PMATH_API 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.

7.3.2.21 PMATH_API 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)

7.3.2.22 PMATH_API 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

7.4 Objects - the Base of pMath

The basic class for all pMath objects.

Data Structures

- class pmath_t

    The basic type of all pMath objects.

Defines

- #define PMATH_IS_MAGIC(obj)

    Fast test, whether an object is a 'magic value'.

- pmath_t::PMATH_IS_MAGIC(obj)

    Fast test, whether an object is a 'magic value'.

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.

Functions

- PMATH_API unsigned int pmath_t::pmath_hash (pmath_t obj)

    Calculates an object's hash value.

- PMATH_API pmath_bool_t pmath_t::pmath_equals (pmath_t objA, pmath_t objB)

    Compares two objects for identity.

- PMATH_API int pmath_t::pmath_compare (pmath_t objA, pmath_t objB)

  Compares two objects syntactically.

- PMATH_API void pmath_t::pmath_write (pmath_t obj, pmath_write_options_t options, pmath_write_func_t write, void ∗user)

  Write an object to a stream.

- PMATH_API pmath_bool_t pmath_t::pmath_is_evaluated (pmath_t obj)

  Test whether an object is already evaluated.

7.4.1 Detailed Description

The basic class for all pMath objects.

pMath works on objects. They can be expressions (trees of pMath objects), symbols, values, strings or 'magic objects' (integer value between 0 and 255).

See also:

Object Utility Functions

7.4.2 Define Documentation

7.4.2.1 #define PMATH_IS_MAGIC(obj)

Fast test, whether an object is a 'magic value'.

Parameters:

obj A pMath object

Returns:

A boolean value.

This is the faster equivalent to pmath_instance_of(obj, PMATH_TYPE_MAGIC).

See also:

pmath_instance_of()

7.4.2.2 PMATH_IS_MAGIC(obj) `[inherited]`

Fast test, whether an object is a 'magic value'.

Parameters:

obj A pMath object

Returns:

A boolean value.

This is the faster equivalent to pmath_instance_of(obj, PMATH_TYPE_MAGIC).

See also:

pmath_instance_of()

7.4.3 Typedef Documentation

7.4.3.1 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_MAGIC` PMATH_TYPE_MAGIC: The object is a ‘magic number‘, which is any integer value between 0 and 255 cast to (pmath_t). These values have special meanings:
  - NULL This is simply ‘nothing‘. It is often used to indicate that there is not enough memory.
  - PMATH_UNDEFINED PMATH_UNDEFINED Symbol values are initialized with PMATH_UNDEFINED. This is done to enable saving the value NULL in a symbol.

Any function that returns or operates on a pmath_t may return NULL. Exceptions are allways explicitly stated in the documentation.

- `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. Because of this, you almost never test for this type, but for PMATH_TYPE_RATIONAL, since the result of an operation on two quotients may be an integer. You can cast quotient objects to pmath_rational_t and thus to pmath_number_t too.

- `PMATH_TYPE_RATIONAL`: The object is either an integer or a quotient. You can cast it to pmath_rational_t and thus to pmath_number_t too.

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

- `PMATH_TYPE_FLOAT`: The object is either PMATH_TYPE_MP_FLOAT or PMATH_TYPE_MACHINE_FLOAT.

- `PMATH_TYPE_NUMBER`: The object is a numerical value (integer, quotient, floating point number). You can cast it to pmath_number_t.

Note that complex numbers are stored as expressions and thus are not numbers in this sense. Additionally, algebraic and special constants as Sqrt(2) or Pi are not numbers, but symbols or expressiones (e.g. Sqrt(2)).

- `PMATH_TYPE_STRING`: 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_EXPRESSION`: The object is an expression. You can cast it to pmath_expr_t.
- `PMATH_TYPE_CUSTOM`: The object is a custom object. You can cast it to pmath_custom_t.
- `PMATH_TYPE_EVALUATABLE`: The object is evaluatable. That means, if a symbol has this object as its value, the object will be returned. Function definition rules and custom objects are an example of non-evalutable objects.

7.4.3.2 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.

7.4.3.3 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.

7.4.3.4 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.

7.4.3.5 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.

7.4.3.6 typedef unsigned int(∗ pmath_hash_func_t)(pmath_t)

A hash function for an object.

If two objects equal, their hash values equal.

7.4.3.7 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.

7.4.3.8 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.

7.4.4 Function Documentation

7.4.4.1 PMATH_API 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).

7.4.4.2 PMATH_API pmath_bool_t pmath_equals (pmath_t *objA*, pmath_t *objB*) `[inherited]`

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 q floating point value B.

7.4.4.3 PMATH_API 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.

7.4.4.4 PMATH_API void pmath_write (pmath_t *obj*, pmath_write_options_t *options*, pmath_write_func_t *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

7.4.4.5 PMATH_API 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.

7.5 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(NULL, 0, (cstr), -1)

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

- pmath_string_t::PMATH_C_STRING(cstr)

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

Functions

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

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

- PMATH_API 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.*

- PMATH_API uint32_t pmath_char_from_name (const char ∗name)

  Get a named character.

- PMATH_API const char ∗ pmath_char_to_name (uint32_t unichar)

  Get a character's name.

- PMATH_API pmath_string_t pmath_string_t::pmath_string_new (int capacity)

  Create an empty pMath String.

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

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

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

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

- PMATH_API pmath_string_t 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.

- PMATH_API char ∗ pmath_string_t::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_API pmath_string_t 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_API pmath_string_t 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_API pmath_string_t 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_API pmath_string_t pmath_string_t::pmath_string_concat (pmath_string_t prefix, pmath_string_t postfix)

  Concatenate two pMath Strings.

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

  Extract a substring of a pMath String.

- PMATH_API const uint16_t ∗ pmath_string_t::pmath_string_buffer (pmath_string_t string)

  Get a string's buffer for reading.

- PMATH_API int pmath_string_t::pmath_string_length (pmath_string_t string)

Get a string's length.

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

  Compare a pMth sting with a C string.

7.5.1 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.

7.5.2 Define Documentation

7.5.2.1 #define PMATH_C_STRING(cstr) pmath_string_insert_latin1(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().

7.5.2.2 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().

7.5.3 Function Documentation

7.5.3.1 PMATH_API 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);
```

7.5.3.2 PMATH_API 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().

7.5.3.3 PMATH_API 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

7.5.3.4 PMATH_API 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

7.5.3.5 PMATH_API 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 NULL on failure. You must destroy it.

7.5.3.6 PMATH_API 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 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:

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

If `str` is 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.

7.5.3.7 PMATH_API 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:

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

7.5.3.8 PMATH_API 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 NULL on error. You have to free the memory with pmath_mem_free(result, ∗size_ptr).

Note:

pMath strings may contain embedded ", 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 ".

7.5.3.9 PMATH_API 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:

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

7.5.3.10 PMATH_API 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 NULL on error. You have to free the memory with pmath_mem_free(result, ∗size_ptr).

Note:

pMath strings may contain embedded ”, 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 ”.

7.5.3.11 PMATH_API 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 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:

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

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

7.5.3.12 PMATH_API 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 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:

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

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

7.5.3.13 PMATH_API 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 NULL. It will be freed.

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

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

Returns:

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

7.5.3.14 PMATH_API 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:

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 NULL, the other string will be returned.

7.5.3.15 PMATH_API 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:

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`.

7.5.3.16 PMATH_API const uint16_t * pmath_string_buffer (pmath_string_t *string*) [inherited]

Get a string's buffer for reading.

Parameters:

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

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.

7.5.3.17 PMATH_API 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.

7.5.3.18 PMATH_API pmath_bool_t pmath_string_equals_latin1 (pmath_string_t *string*, const char * *latin1*) [inherited]

Compare a pMth sting 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);
```

7.6 Symbols

Symbol objects in pMath.

Data Structures

- class pmath_symbol_t

  The Symbol class.

Typedefs

- typedef pmath_t(∗ pmath_builtin_func_t)(pmath_expr_t expr)

  Prototype of a built-in function.

- typedef int pmath_symbol_attributes_t

  The (bitset) type of symbol attributes.

Enumerations

- enum pmath_code_usage_t

  Constants saying when to call a binded function. May f be the pMath symbol.

Functions

- PMATH_API pmath_symbol_t pmath_symbol_builtin (int index)

  Get a builtin symbol object.

- PMATH_API pmath_bool_t pmath_register_code (pmath_symbol_t symbol, pmath_builtin_func_t func, pmath_code_usage_t usage)

  Bind a pMath symbol to some native code.

- PMATH_API pmath_bool_t pmath_register_approx_code (pmath_symbol_t symbol, pmath_t(∗func)(pmath_t, double, double))

  Bind a pMath symbol to some native code for approximation.

- PMATH_API void pmath_symbol_remove (pmath_symbol_t symbol)

  Remove a symbol completely from the system.

- PMATH_API pmath_symbol_t pmath_symbol_iter_next (pmath_symbol_t old)

  Iterate through the global symbol table.

- PMATH_API pmath_symbol_t pmath_symbol_t::pmath_symbol_get (pmath_string_t name, pmath_bool_t create)

  Get a symbol by its fully qualified name.

- PMATH_API pmath_symbol_t pmath_symbol_t::pmath_symbol_create_temporary (pmath_string_t name, pmath_bool_t unique)

  Create a new temporary symbol.

- PMATH_API pmath_symbol_t pmath_symbol_t::pmath_symbol_find (pmath_string_t name, pmath_bool_t create)

  Find a symbol in the current namespace search path.

- PMATH_API pmath_string_t pmath_symbol_t::pmath_symbol_name (pmath_symbol_t symbol)

  Get a symbol's name.

- PMATH_API pmath_symbol_attributes_t pmath_symbol_t::pmath_symbol_get_attributes (pmath_symbol_t symbol)

  Get a symbol's attributes.

- PMATH_API void pmath_symbol_t::pmath_symbol_set_attributes (pmath_symbol_t symbol, pmath_symbol_attributes_t attr)

  Set a symbol's attributes.

- PMATH_API pmath_t pmath_symbol_t::pmath_symbol_get_value (pmath_symbol_t symbol)

  Get a symbol's value.

- PMATH_API void pmath_symbol_t::pmath_symbol_set_value (pmath_symbol_t symbol, pmath_t value)

  Set a symbol's value.

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

  Execute a function synchronized to a symbol.

- PMATH_API void pmath_symbol_t::pmath_symbol_update (pmath_symbol_t symbol)

  Update a symbol manually.

7.6.1 Detailed Description

Symbol objects in pMath.

7.6.2 Typedef Documentation

7.6.2.1 typedef pmath_t(∗ pmath_builtin_func_t)(pmath_expr_t expr)

Prototype of a built-in function.

Parameters:

expr An expression. The function must free it.

Returns:

Any pMath object.

7.6.2.2 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.

7.6.3 Enumeration Type Documentation

7.6.3.1 enum pmath_code_usage_t

Constants saying when to call a binded function. May f be the pMath symbol.

- PMATH_CODE_USAGE_DOWNCALL Call binded function when evaluating 'f(...)'
- PMATH_CODE_USAGE_SUBCALL Call binded function when evaluating 'f(...)(...)...'
- PMATH_CODE_USAGE_UPCALL Call binded function when evaluating 'g(...,f,...)' or 'g(..., f(...), ...)'

These use cases correspond to the DownRules(), SubRules() and UpRules() of a symbol.

See also:

pmath_register_code

7.6.4 Function Documentation

7.6.4.1 PMATH_API pmath_symbol_t pmath_symbol_builtin (int *index*)

Get a builtin symbol object.

Parameters:

index The symbols unique index.

Returns:

The builtin symbol. This is not a copy. You must not destroy it with e.g. pmath_unref().

When you want to use the returned symbol, pmath_ref() it first. Normaly, you do not call this function directly but use one of the PMATH_SYMBOL_XXX macros instead, where XXX is the uppercase equivalent of the corresponding pMath symbol name, e.g. PMATH_SYMBOL_LIST.

7.6.4.2 PMATH_API pmath_bool_t pmath_register_code (pmath_symbol_t *symbol*, pmath_builtin_func_t *func*, pmath_code_usage_t *usage*)

Bind a pMath symbol to some native code.

Parameters:

symbol The symbol. It wont be freed.

func A native function.

usage When to call the native function.

Returns:

Whether the binding succeded.

Any previous binding to the symbol with the same *usage* will be overwritten.

7.6.4.3 PMATH_API pmath_bool_t pmath_register_approx_code (pmath_symbol_t *symbol*, pmath_t(∗)(pmath_t, double, double) *func*)

Bind a pMath symbol to some native code for approximation.

Parameters:

symbol The symbol. It wont be freed.

func A native function.

Returns:

Whether the binding succeded.

Any previous approximation-binding to the symbol will be overwritten.

The native function will be calledd when evaluating the pMath expression `N(obj, {precision, accuracy})` as `func(obj, precision, accuracy)`, where obj is either ‘symbol‘ or ‘symbol(...)‘.

7.6.4.4 PMATH_API void pmath_symbol_remove (pmath_symbol_t *symbol*)

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.

7.6.4.5 PMATH_API pmath_symbol_t pmath_symbol_iter_next (pmath_symbol_t *old*)

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 && iter != PMATH_SYMBOL_LIST);
pmath_unref(iter);
```

7.6.4.6 PMATH_API 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:

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

7.6.4.7 PMATH_API 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.

7.6.4.8 PMATH_API 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:

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

7.6.4.9 PMATH_API 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().

7.6.4.10 PMATH_API 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.

7.6.4.11 PMATH_API 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.

7.6.4.12 PMATH_API 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).

7.6.4.13 PMATH_API 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.

7.6.4.14 PMATH_API 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

7.6.4.15 PMATH_API void pmath_symbol_update (pmath_symbol_t *symbol*) `[inherited]`

Update a symbol manually.

Parameters:

symbol A pMath symbol.

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.

7.7 C++ Binding

7.7.1 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.

Todo

Document the Expr helper functions.

7.8 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.

Enumerations

- enum pmath_token_t

  Token classes known in the pMath language.

Functions

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

  Analyse a token.

- PMATH_API 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.

- PMATH_API void pmath_span_array_t::pmath_span_array_free (pmath_span_array_t ∗spans)

  Destroy a span-array and all its spans.

- PMATH_API int pmath_span_array_t::pmath_span_array_length (pmath_span_array_t ∗spans)

  Get a span-array's length.

- PMATH_API pmath_bool_t pmath_span_array_t::pmath_span_array_is_token_end (pmath_span_array_t ∗spans, int pos)

  Test the token-end-flag at an index.

- PMATH_API pmath_bool_t pmath_span_array_t::pmath_span_array_is_operator_start (pmath_span_array_t ∗spans, int pos)

  Test the operator-start-flag at an index.

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

  Get a span starting at an index.

- PMATH_API pmath_span_t ∗ pmath_span_t::pmath_span_next (pmath_span_t ∗span)

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

- PMATH_API int pmath_span_t::pmath_span_end (pmath_span_t ∗span)

  Get end of a span.

- PMATH_API pmath_t pmath_string_t::pmath_string_expand_boxes (pmath_string_t s)

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

- PMATH_API pmath_t pmath_string_t::pmath_parse_string (pmath_string_t code)

  Parse a string to an expression.

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

  Parse a string with additional arguments to an expression.

7.8.1 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.

7.8.2 Example

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

`{UnderoverscriptBox("\u2211", {"i", "=", "1"}, "n"), {"f", "(", "i", ")"}}`

or

`{{"\u2211", SubsuperscriptBox({"i", "=", "1"}, "n")}, {"f", "(", "i", ")"}}`

(U+2211 is the Unicode character "N-ARY SUMMATION").

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

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

7.8.3 Define Documentation

7.8.3.1 #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).

7.8.3.2 #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 ...

7.8.4 Function Documentation

7.8.4.1 PMATH_API 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 A function to be called, when there is more input needed. Its result will be appended to ∗code.

subsuperscriptbox_at_index A 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 Iff there is an UnderscriptBox, OverscriptBox or UnderoverscriptBox at a given position in the code (indicated by the PMATH_CHAR_BOX character) its base (middle part of UnderoverscriptBox) should be returned by this function, otherwise 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 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‘BoxesToExpression(). The span-array must be freed with pmath_span_array_free() when it is no longer needed.

7.8.4.2 PMATH_API 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 A 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.

7.8.4.3 PMATH_API 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.

7.8.4.4 PMATH_API 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.

7.8.4.5 PMATH_API 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.

7.8.4.6 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

7.8.4.7 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.

7.8.4.8 PMATH_API void pmath_span_array_free (pmath_span_array_t ∗ *spans*) `[inherited]`

Destroy a span-array and all its spans.

Parameters:

spans The span-array.

7.8.4.9 PMATH_API 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 NULL.

7.8.4.10 PMATH_API 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.

7.8.4.11 PMATH_API pmath_bool_t pmath_span_array_is_operator_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.

7.8.4.12 PMATH_API 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 NULL if there is no span.

7.8.4.13 PMATH_API 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 NULL if there is no span.

7.8.4.14 PMATH_API 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.

7.8.4.15 PMATH_API 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.

7.8.4.16 PMATH_API 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 Release(BoxesToExpression(StringToBoxes("code"))), but does not evaluate this released result.

7.8.4.17 PMATH_API 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 before
4. returning the result of the pmath_parse_string-call.

See also:

pmath_build_value
pmath_parse_string

7.9 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.

Typedefs

- typedef char pmath_bool_t

  A boolean type.

- typedef void(∗ pmath_callback_t)(void ∗)

  A general callback function.

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

  A UCS-2 writer callback function.

- typedef void(∗ pmath_write_unichar_func_t)(void ∗user, uint32_t ch)

  A callback function to write a single unicode character.

7.9.1 Detailed Description

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

7.9.2 Typedef Documentation

7.9.2.1 typedef char pmath_bool_t

A boolean type.

C does not define a boolean type, so we do it 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 FALSE; instead of return 0;

7.9.2.2 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.

7.9.2.3 typedef void(∗ pmath_write_func_t)(void ∗user, const uint16_t ∗data, int len)

A UCS-2 writer callback function.

The to-be-written data is not zero-terminated.

7.10 Atomic Operations

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

Functions

- intptr_t pmath_atomic_fetch_add (intptr_t volatile ∗atom, intptr_t delta)

  Add a value to another.

- intptr_t pmath_atomic_fetch_set (intptr_t volatile ∗atom, intptr_t new_value)

  Exchange a value.

- intptr_t pmath_atomic_fetch_compare_and_set (intptr_t volatile ∗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 (intptr_t volatile ∗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 (intptr_t volatile ∗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 (intptr_t volatile ∗atom)

  Try to aquire a lock.

- void pmath_atomic_unlock (intptr_t volatile ∗atom)

  Release a previously aquired lock.

- void pmath_atomic_loop_nop (void)

7.10.1 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.

7.10.2 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

7.10.3 Function Documentation

7.10.3.1 intptr_t pmath_atomic_fetch_add (intptr_t volatile ∗ *atom*, intptr_t *delta*)

Add a value to another.

Parameters:

atom A sizeof(void∗) aligned pointer.

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.

7.10.3.2 intptr_t pmath_atomic_fetch_set (intptr_t volatile ∗ *atom*, intptr_t *new_value*)

Exchange a value.

Parameters:

atom A sizeof(void∗) aligned pointer.

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.

7.10.3.3 intptr_t pmath_atomic_fetch_compare_and_set (intptr_t volatile ∗ *atom*, intptr_t *old_value*, intptr_t *new_value*)

Exchange a value if it equals another value.

Parameters:

atom A sizeof(void∗) aligned pointer.

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.

7.10.3.4 pmath_bool_t pmath_atomic_compare_and_set (intptr_t volatile ∗ *atom*, intptr_t *old_value*, intptr_t *new_value*)

Exchange a value if it equals another value.

Parameters:

atom A sizeof(void∗) aligned pointer.

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.

7.10.3.5 pmath_bool_t pmath_atomic_compare_and_set_2 (intptr_t volatile ∗ *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*** A 2∗sizeof(void∗) aligned pointer to two values.

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.

7.10.3.6 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).

7.10.3.7 void pmath_atomic_lock (intptr_t volatile ∗ *atom*)

Try to aquire a lock.

Parameters:

atom The lock. A sizeof(void∗) aligned pointer.

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

```
PMATH_DECLARE_ATOMIC(spin);
...
pmath_atomic_lock(&spin)
... critical section ...
pmath_atomic_unlock(&spin);
```

7.10.3.8 void pmath_atomic_unlock (intptr_t volatile ∗ *atom*)

Release a previously aquired lock.

Parameters:

atom The lock. A sizeof(void∗) aligned pointer.

See also:

pmath_atomic_lock

7.10.3.9 void pmath_atomic_loop_nop (void)

A no-operation for use in spin locks.

7.11 Thread Messaging

Sending messages to other threads.

Data Structures

- class pmath_messages_t

  A message queue for interthread communication.

Functions

- PMATH_API pmath_bool_t pmath_messages_t::pmath_is_message_queue (pmath_t obj)

  Test if an object is a message queue.

- PMATH_API pmath_messages_t pmath_messages_t::pmath_thread_get_queue (void)

  Get the current thread's message queue.

- PMATH_API void pmath_messages_t::pmath_thread_sleep (void)

  Send the current thread to sleep.

- PMATH_API void pmath_messages_t::pmath_thread_wakeup (pmath_messages_t mq)

  Wake up another thread.

- PMATH_API void pmath_messages_t::pmath_thread_send (pmath_messages_t mq, pmath_t msg)

  Asynchronously send a message to another thread.

- PMATH_API pmath_t pmath_messages_t::pmath_thread_send_wait (pmath_messages_t mq, pmath_t msg, double timeout_seconds)

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

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

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

7.11.1 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 any 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 signal handler.

7.11.2 Function Documentation

7.11.2.1 PMATH_API 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).

7.11.2.2 PMATH_API 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 NULL on error. You must destroy it with pmath_unref() when its no longer needed.

7.11.2.3 PMATH_API 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.

7.11.2.4 PMATH_API 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.

7.11.2.5 PMATH_API 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.

7.11.2.6 PMATH_API pmath_t pmath_thread_send_wait (pmath_messages_t *mq*, pmath_t *msg*, double *timeout_seconds*) `[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.

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)

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.

7.11.2.7 PMATH_API 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.

7.12 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.

Functions

- PMATH_API pmath_t pmath_thread_local_save (pmath_t key, pmath_t value)

  Store a thread/thread-local value.

- PMATH_API pmath_t pmath_thread_local_load (pmath_t key)

  Load a thread/thread-local value.

- PMATH_API void pmath_throw (pmath_t exception)

  Throw an exception.

- PMATH_API pmath_t pmath_catch (void)

  Catch any exception.

- PMATH_API pmath_bool_t pmath_aborting (void)

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

- PMATH_API void pmath_abort_please (void)

  Requests pMath to abort the current evaluation.

- PMATH_API 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().

- PMATH_API void pmath_resume_all (void)

  Resume all other threads.

- PMATH_API void pmath_threadlock_t::pmath_thread_call_locked (pmath_threadlock_t ∗threadlock_ptr, pmath_callback_t callback, void ∗data)

  Execute a function synchronized with a threadlock.

- PMATH_API pmath_thread_t pmath_thread_t::pmath_thread_get_current (void)

  Get the current pMath thread.

- PMATH_API pmath_thread_t pmath_thread_t::pmath_thread_get_parent (pmath_thread_t thread)

  Get a thread's direct parent.

- PMATH_API pmath_bool_t pmath_thread_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_API pmath_bool_t pmath_thread_t::pmath_thread_aborting (pmath_thread_t thread)

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

7.12.1 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.

7.12.2 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.

7.12.3 Function Documentation

7.12.3.1 PMATH_API 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 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).

7.12.3.2 PMATH_API 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.

7.12.3.3 PMATH_API 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.

7.12.3.4 PMATH_API 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().

7.12.3.5 PMATH_API 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.

7.12.3.6 PMATH_API void pmath_abort_please (void)

Requests pMath to abort the current evaluation.

This function is signal-safe.

See also:

pmath_continue_after_abort()

7.12.3.7 PMATH_API void pmath_resume_all (void)

Resume all other threads.

See also:

pmath_suspend_all_please

7.12.3.8 PMATH_API 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().

7.12.3.9 PMATH_API pmath_thread_t pmath_thread_get_current (void) `[related, inherited]`

Get the current pMath thread.

Returns:

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

7.12.3.10 PMATH_API 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 NULL.

7.12.3.11 PMATH_API 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.

7.12.3.12 PMATH_API 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

7.13 Debugging

Functions

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

  Print out a simple debug message.

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

  Print a pMath object to the debug log.

7.13.1 Detailed Description

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

7.13.2 Function Documentation

7.13.2.1 PMATH_API 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.

7.13.2.2 PMATH_API 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.

7.14 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_API pmath_bool_t pmath_file_test (pmath_t file, int properties)

  Check whether a file supports a set of properties.

- PMATH_API pmath_files_status_t pmath_file_status (pmath_t file)

  Get the current status of a readable file.

- PMATH_API 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_API pmath_string_t pmath_file_readline (pmath_t file)

  Read one line from a text file.

- PMATH_API void pmath_file_set_textbuffer (pmath_t file, pmath_string_t buffer)

  Set a file's text buffer.

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

  Write some bytes to a binary file.

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

  Write to a text file.

- PMATH_API void pmath_file_flush (pmath_t file)

  Flush the output buffer of a writeable file.

- PMATH_API 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_API pmath_bool_t pmath_file_set_binbuffer (pmath_t file, size_t size)

  Set a binary file's buffer size.

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

  Manipulate a file's internal representation.

- PMATH_API pmath_bool_t pmath_file_close (pmath_t file)

  Closes a file.

- PMATH_API 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_API 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_API 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_API pmath_symbol_t pmath_file_create_binary_buffer (size_t mincapacity)

  Create a byte-stream file object.

- PMATH_API size_t pmath_file_binary_buffer_size (pmath_t binfile)

  Get The number of readable bytes in a binary buffer.

- PMATH_API 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.

7.14.1 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.

7.14.2 Enumeration Type Documentation

7.14.2.1 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.

7.14.3 Function Documentation

7.14.3.1 PMATH_API 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.

7.14.3.2 PMATH_API 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.

7.14.3.3 PMATH_API 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*.

7.14.3.4 PMATH_API 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.

7.14.3.5 PMATH_API void pmath_file_set_textbuffer (pmath_t *file*, pmath_string_t *buffer*)

Set a file's text buffer.

Parameters:

file A readable text file. It wont be freed.

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

7.14.3.6 PMATH_API 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.

7.14.3.7 PMATH_API 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.

7.14.3.8 PMATH_API 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.

7.14.3.9 PMATH_API 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.

7.14.3.10 PMATH_API 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.

7.14.3.11 PMATH_API 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.

7.14.3.12 PMATH_API 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.

7.14.3.13 PMATH_API 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

7.14.3.14 PMATH_API 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

7.14.3.15 PMATH_API 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().

7.14.3.16 PMATH_API 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.

7.14.3.17 PMATH_API 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.

7.14.3.18 PMATH_API 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*.

7.15 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_API 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_API 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_API pmath_t pmath_current_head (void)

  Get the currently evaluated function.

- PMATH_API void pmath_walk_stack (pmath_stack_walker_t walker, void ∗closure)

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

- PMATH_API pmath_t pmath_session_start (void)

  Saves some global state when an interactive dialog session starts.

- PMATH_API void pmath_session_end (pmath_t old_state)

  Restore some global state when an interactive dialog session ends.

- PMATH_API void pmath_expr_t::pmath_gather_begin (pmath_t pattern)

  Start gathering emitted objects.

- PMATH_API pmath_expr_t pmath_expr_t::pmath_gather_end (void)

  Finish gathering emitted objects.

- PMATH_API void pmath_expr_t::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_API pmath_t pmath_t::pmath_build_value_v (const char ∗format, va_list args)

  Generate a List of objects with a format string.

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

  Generate a List of objects with a format string.

- PMATH_API pmath_expr_t pmath_expr_t::pmath_options_extract (pmath_expr_t expr, size_t last_nonoption)

  Extract custom option values from an expression.

- PMATH_API pmath_t pmath_expr_t::pmath_option_value (pmath_t fn, pmath_t name, pmath_t extra)

  Retrieve a option value of a given function.

7.15.1 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.

7.15.2 Typedef Documentation

7.15.2.1 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.

7.15.3 Function Documentation

7.15.3.1 PMATH_API 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* which must be a symbol.

7.15.3.2 PMATH_API 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* which must be a symbol.

7.15.3.3 PMATH_API 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.

7.15.3.4 PMATH_API 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.

7.15.3.5 PMATH_API 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.

7.15.3.6 PMATH_API 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.

7.15.3.7 PMATH_API 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

7.15.3.8 PMATH_API 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

7.15.3.9 PMATH_API 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

7.15.3.10 PMATH_API 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.

This function interface is only given if stdarg.h was included before.

See also:

pmath_build_value

7.15.3.11 PMATH_API 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 Ininity are converted to Indeterminate and +/-Intinity
- 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∗] 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. Windows but not 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. Windows but not Linux.
- Ctt [maching the 2 t's] build a complex value
- Qtt [maching the 2 t's] build a rational value from two integers.
- (items) [matching items] constructs a sublist of items.

Note:

When the format string denotes only one object, this object wille 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.

7.15.3.12 PMATH_API pmath_expr_t pmath_options_extract (pmath_expr_t *expr*, size_t *last_nonoption*) [related, inherited]

Extract custom option values from an expression.

Parameters:

expr The expression that may custom option values. It wont be freed.

last_nonoption The index of the last argument that is not a option rule.

Returns:

A list of all given option values or NULL on error. You must destroy it.

Imagine, `expr` = ‘f(a,b,A->1,B->2)‘ and `last_nonoption` is 2, then the result value is a list ‘{A->1, B->2}‘. You can use this return value as the `extra` parameter in pmath_option_value().

When `last_nonoption` was 1, a message would be generated (b is no rule ...) and the return value is NULL. In that case, the calling function should have no further effects and return.

7.15.3.13 PMATH_API 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 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.

7.16 Memory Management

Memory management for pMath.

Functions

- PMATH_API void ∗ pmath_mem_alloc (size_t size)

  Allocate some amount of memory.

- PMATH_API void ∗ pmath_mem_realloc (void ∗p, size_t new_size)

  Change the size of a memory-chunk.

- PMATH_API void ∗ pmath_mem_realloc_no_failfree (void ∗p, size_t new_size)

  Change the size of a memory-chunk.

- PMATH_API void pmath_mem_free (void ∗p)

  Free a memory-chunk.

- PMATH_API void pmath_mem_usage (size_t ∗current, size_t ∗max)

  Get memory usage information.

7.16.1 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 (a simple garbage collector for pMath symbols and numbers, which are not freed automatically, starts).

See also:

PMATH_INIT_WINDOWS_DONT_USE_LOW_FRAGMENTATION_HEAP

7.16.2 Function Documentation

7.16.2.1 PMATH_API 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().

7.16.2.2 PMATH_API 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 result is NULL because there is not enough memory.

You must free the result with pmath_mem_free() or indirectly via pmath_mem_realloc().

7.16.2.3 PMATH_API 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.

7.16.2.4 PMATH_API 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().

7.16.2.5 PMATH_API 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 until now.

7.17 Messages

Error handling and informing the user.

Functions

- PMATH_API void pmath_message (pmath_symbol_t symbol, const char ∗tag, size_t argcount,...)

  Print a message with pMath object arguments.

- PMATH_API void pmath_message_argxxx (size_t given, size_t min, size_t max)

  Generate a General::arg∗ message (invalid argument count).

- PMATH_API pmath_string_t pmath_message_find_text (pmath_t name)

    Find a message's text.

- PMATH_API 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.

7.17.1 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 Power function invokes the Power::indet in case of 0^0.

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.

7.17.2 Function Documentation

7.17.2.1 PMATH_API 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. NULL will be treated as pmath_current_head(). This is useful, because pmath_current_head() returns a reference, but pmath_message() would not free it.

tag The message's tag as a zero-terminated C string.

argcount The number of following arguments.

... Exactly *argcount* pMath objects. They all will be freed.

Note:

If symbol==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.

7.17.2.2 PMATH_API 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.

7.17.2.3 PMATH_API 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 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.

7.17.2.4 PMATH_API 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 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 NULL.

This function poduces 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 not NULL. It can be used to report errors/warnings from pmath_spans_from_string().

7.18 Threadsafe Stacks

Fast threadsafe stacks in pMath.

Data Structures

- class pmath_stack_t

  The type of pMath's threadsafe stacks.

Functions

- PMATH_API pmath_stack_t pmath_stack_new (void)

  Create an empty stack.

- PMATH_API void pmath_stack_free (pmath_stack_t stack)

  Destroy a stack.

- PMATH_API void pmath_stack_push (pmath_stack_t stack, void ∗item)

  Push an item onto a stack.

- PMATH_API void ∗ pmath_stack_pop (pmath_stack_t stack)

  Pop an item from a stack.

7.18.1 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://pagesperso-orange.fr/gmem/evenements/jim2002/articles/L17_Fober.pdf)

7.18.2 Function Documentation

7.18.2.1 PMATH_API pmath_stack_t pmath_stack_new (void)

Create an empty stack.

Returns:

A new stack or NULL.

Note that you cannot push anything onto a NULL stack, so check the result. Free the result with pmath_stack_free().

7.18.2.2 PMATH_API void pmath_stack_free (pmath_stack_t *stack*)

Destroy a stack.

Parameters:

stack A stack previously created with pmath_stack_new(). May be NULL.

You must manually pop and free all items on the stack before calling this function, because those items would not be freed automatically.

7.18.2.3 PMATH_API 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 NULL.

item The item to be pushed. It must point to a structure whose first element is a pointer. Must not be NULL.

7.18.2.4 PMATH_API 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 NULL.

Returns:

The top of stack or NULL if it is empty.

7.19 Front-ends

Functions for use front-ends.

Functions

- PMATH_API pmath_bool_t pmath_continue_after_abort (void)

  Requests pMath to stop aborting the current evaluation.

- PMATH_API pmath_t pmath_session_execute (pmath_t input, pmath_bool_t ∗aborted)

  Execute an expression and change $History and $Line appropriately.

- PMATH_API pmath_bool_t pmath_init (void)

Initialize the pMath CAS library.

- PMATH_API void pmath_done (void)

  Free all resources used by the pMath CAS library and unload all modules.

7.19.1 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.

7.19.2 Function Documentation

7.19.2.1 PMATH_API 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()

7.19.2.2 PMATH_API 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.

7.19.2.3 PMATH_API 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.

7.19.2.4 PMATH_API void pmath_done (void)

Free all resources used by the pMath CAS library and unload all modules.

See also:

pmath_init

8 Data Structure Documentation

8.1 pmath::Expr Class Reference

A wrapper for pmath_t and drived types.

Inherited by pmath::String.

Public Member Functions

- Expr (pmath_t obj=0) throw ()

  Construct form a pmath_t, stealing the reference.

- bool instance_of (pmath_type_t type) const throw ()

  Check for underlying pMath C API type.

- 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 identity. The pMath =!= operator.

- 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 null pointer.

- size_t expr_length () const throw ()

  Length of the expression of 0 on error.

- Expr operator[ ] (size_t i) const throw ()

  i-th argument of the expression.

- 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.

8.1.1 Detailed Description

A wrapper for pmath_t and drived types.

This class wraps a single pmath_t, so its size is sizeof(void∗).

8.1.2 Member Function Documentation

8.1.2.1 Expr pmath::Expr::operator[ ] (size_t *i*) const throw () `[inline]`

i-th argument of the expression.

Parameters:

i Index. May be > 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.

8.1.2.2 double pmath::Expr::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.

8.1.2.3 String pmath::Expr::to_string (pmath_write_options_t *options* = 0) const throw () [inline]

Convert to a string.

Parameters:

options Optional output options.

Returns:

The String representation.

The documentation for this class was generated from the following file:

- pmath-cpp.h

8.2 pmath::Gather Class Reference

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.

Public Member Functions

- Expr end ()

  end gathering.

Static Public Member Functions

- static void Emit (Expr e)

  emit a value to be gathered.

8.2.1 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.

Note:

Gathering multiple lists at once is not supported!

The documentation for this class was generated from the following file:

- pmath-cpp.h

8.3 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 NULL if read_function is also 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.

8.3.1 Detailed Description

Access functions for binary files.

See also:

pmath_file_create_binary

The documentation for this class was generated from the following file:

- pmath-util/files.h

8.4 pmath_cstr_writer_info_t Struct Reference

Additional information for pmath_utf8_writer() or pmath_native_writer().

8.4.1 Detailed Description

Additional information for pmath_utf8_writer() or pmath_native_writer().

The documentation for this struct was generated from the following file:

- pmath-core/strings.h

8.5 pmath_custom_t Class Reference

The Custom Object class.

Inherits pmath_t.

Inherited by pmath_messages_t.

Public Member Functions

- PMATH_API pmath_custom_t pmath_custom_new (void ∗data, pmath_callback_t destructor)

  Create a custom object.

- PMATH_API void ∗ pmath_custom_get_data (pmath_custom_t custom)

  Get a custom object's data member.

- PMATH_API pmath_bool_t pmath_custom_has_destructor (pmath_custom_t custom, pmath_callback_t dtor)

  Get a custom object's data destructor.

8.5.1 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

8.6 pmath_expr_t Class Reference

The Expression class.

Inherits pmath_t.

Public Member Functions

- PMATH_API pmath_expr_t pmath_expr_new (pmath_t head, size_t length)

  Create a new expression.

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

  Create a new expression with all items given.

- PMATH_API pmath_expr_t pmath_expr_resize (pmath_expr_t expr, size_t new_length)

  Resize an expression.

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

  Append some items to an expression.

- PMATH_API size_t pmath_expr_length (pmath_expr_t expr)

Get an expression's length.

- PMATH_API pmath_t pmath_expr_get_item (pmath_expr_t expr, size_t index)

  Get an item from an expression.

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

  Get multiple items from an expression.

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

  Set an item in an expression.

- PMATH_API pmath_expr_t pmath_expr_remove_all (pmath_expr_t expr, pmath_t rem)

  Remove all occurencies of an object from an expression.

- PMATH_API pmath_expr_t pmath_expr_sort (pmath_expr_t expr)

  Sort an expression.

- PMATH_API 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.)

- PMATH_API pmath_t pmath_evaluate_expression (pmath_expr_t expr, pmath_bool_t apply_rules)

  Partly evaluate an expression.

- PMATH_API void pmath_gather_begin (pmath_t pattern)

  Start gathering emitted objects.

- PMATH_API pmath_expr_t pmath_gather_end (void)

  Finish gathering emitted objects.

- PMATH_API 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_API pmath_expr_t pmath_options_extract (pmath_expr_t expr, size_t last_nonoption)

  Extract custom option values from an expression.

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

  Retrieve a option value of a given function.

8.6.1 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 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()'.

8.6.2 Friends And Related Function Documentation

8.6.2.1 PMATH_API 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/evaluation.h
- pmath-util/helpers.h

8.7 pmath_float_t Class Reference

The Floating Point Number class.

Inherits pmath_number_t.

Public Member Functions

- PMATH_API 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_API pmath_float_t pmath_float_new_d (double dbl)

  Create a machine-precision floating point number from a double.

8.7.1 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

8.8 pmath_integer_t Class Reference

The Integer class.

Inherits pmath_rational_t.

Public Member Functions

- PMATH_API pmath_integer_t pmath_integer_new_si (signed long int si)

  Create an integer object from a signed long.

- PMATH_API pmath_integer_t pmath_integer_new_ui (unsigned long int ui)

  Create an integer object from an unsigned long.

- PMATH_API pmath_integer_t pmath_integer_new_size (size_t size)

  Create an integer object from an size_t.

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

  Create an integer object from a C String.

- PMATH_API pmath_bool_t pmath_integer_fits_si (pmath_integer_t integer)

  Find out whether a pMath integer fits into a signed long int.

- PMATH_API pmath_bool_t pmath_integer_fits_ui (pmath_integer_t integer)

  Find out whether a pMath integer fits into a unsigned long int.

- PMATH_API signed long int pmath_integer_get_si (pmath_integer_t integer)

Convert a pMath integer to a signed long int.

- PMATH_API unsigned long int pmath_integer_get_ui (pmath_integer_t integer)

  Convert a pMath integer to a unsigned long int.

8.8.1 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

8.9 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_API pmath_bool_t pmath_is_message_queue (pmath_t obj)

  Test if an object is a message queue.

- PMATH_API pmath_messages_t pmath_thread_get_queue (void)

  Get the current thread's message queue.

- PMATH_API void pmath_thread_sleep (void)

  Send the current thread to sleep.

- PMATH_API void pmath_thread_wakeup (pmath_messages_t mq)

  Wake up another thread.

- PMATH_API void pmath_thread_send (pmath_messages_t mq, pmath_t msg)

  Asynchronously send a message to another thread.

- PMATH_API pmath_t pmath_thread_send_wait (pmath_messages_t mq, pmath_t msg, double timeout_seconds)

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

- PMATH_API 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.

8.9.1 Detailed Description

A message queue for interthread communication.

The documentation for this class was generated from the following file:

- pmath-util/concurrency/threadmsg.h

8.10 pmath_number_t Class Reference

The abstract Number class.

Inherits pmath_t.

Inherited by pmath_float_t, and pmath_rational_t.

Public Member Functions

- PMATH_API double pmath_number_get_d (pmath_number_t number)

  Convert a pMath number to a double.

- PMATH_API int pmath_number_sign (pmath_number_t num)

  Get a number's sign.

- PMATH_API pmath_number_t pmath_number_neg (pmath_number_t num)

  Get a number's negative.

8.10.1 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.

The pmath_type_t of numbers is `PMATH_TYPE_NUMBER`.

See also:

Objects - the Base of pMath

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

8.11 pmath_quotient_t Class Reference

The Quotient class.

Inherits pmath_rational_t.

8.11.1 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`. However, you rarely test on this but on `PMATH_TYPE_RATIONAL`, because quotients whose denominator divides the numerator without a remainder, are converted to pmath_integer_t automatically.

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

8.12 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_API pmath_rational_t pmath_rational_new (pmath_integer_t numerator, pmath_integer_t denominator)

  Create a rational number.

- PMATH_API pmath_integer_t pmath_rational_numerator (pmath_rational_t rational)

  Get the numerator of a rational number.

- PMATH_API pmath_integer_t pmath_rational_denominator (pmath_rational_t rational)

  Get the denominator of a rational number.

8.12.1 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 pmath_type_t of rational numbers is `PMATH_TYPE_RATIONAL`.

The documentation for this class was generated from the following file:

- pmath-core/numbers.h

8.13 pmath_span_array_t Class Reference

Internal flat representaion of spans.

Public Member Functions

- PMATH_API void pmath_span_array_free (pmath_span_array_t *spans)

  Destroy a span-array and all its spans.

- PMATH_API int pmath_span_array_length (pmath_span_array_t *spans)

  Get a span-array's length.

- PMATH_API 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_API pmath_bool_t pmath_span_array_is_operator_start (pmath_span_array_t *spans, int pos)

  Test the operator-start-flag at an index.

- PMATH_API pmath_span_t * pmath_span_at (pmath_span_array_t *spans, int pos)

  Get a span starting at an index.

8.13.1 Detailed Description

Internal flat representaion of spans.

The boxform of `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_operator_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:       x   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 can 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().

The documentation for this class was generated from the following file:

- pmath-language/scanner.h

8.14 pmath_span_t Class Reference

Represents a span in a span-array.

Public Member Functions

- PMATH_API pmath_span_t ∗ pmath_span_next (pmath_span_t ∗span)

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

- PMATH_API int pmath_span_end (pmath_span_t ∗span)

  Get end of a span.

8.14.1 Detailed Description

Represents a span in a span-array.

The documentation for this class was generated from the following file:

- pmath-language/scanner.h

8.15 pmath_stack_t Class Reference

The type of pMath's threadsafe stacks.

8.15.1 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

8.16 pmath_string_t Class Reference

The string class.

Inherits pmath_t.

Public Member Functions

- PMATH_API pmath_string_t pmath_string_new (int capacity)

  Create an empty pMath String.

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

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

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

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

- PMATH_API 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.

- PMATH_API 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_API 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_API 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_API 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_API pmath_string_t pmath_string_concat (pmath_string_t prefix, pmath_string_t postfix)

  Concatenate two pMath Strings.

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

  Extract a substring of a pMath String.

- PMATH_API const uint16_t ∗ pmath_string_buffer (pmath_string_t string)

  Get a string's buffer for reading.

- PMATH_API int pmath_string_length (pmath_string_t string)

  Get a string's length.

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

  Compare a pMth sting with a C string.

Related Functions

(Note that these are not member functions.)

- PMATH_API pmath_t pmath_string_expand_boxes (pmath_string_t s)

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

- PMATH_API pmath_t pmath_parse_string (pmath_string_t code)

  Parse a string to an expression.

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

  Parse a string with additional arguments to an expression.

8.16.1 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

8.17 pmath_symbol_t Class Reference

The Symbol class.

Inherits pmath_t.

Public Member Functions

- PMATH_API pmath_symbol_t pmath_symbol_get (pmath_string_t name, pmath_bool_t create)

  Get a symbol by its fully qualified name.

- PMATH_API pmath_symbol_t pmath_symbol_create_temporary (pmath_string_t name, pmath_bool_t unique)

  Create a new temporary symbol.

- PMATH_API pmath_symbol_t pmath_symbol_find (pmath_string_t name, pmath_bool_t create)

  Find a symbol in the current namespace search path.

- PMATH_API pmath_string_t pmath_symbol_name (pmath_symbol_t symbol)

  Get a symbol's name.

- PMATH_API pmath_symbol_attributes_t pmath_symbol_get_attributes (pmath_symbol_t symbol)

  Get a symbol's attributes.

- PMATH_API void pmath_symbol_set_attributes (pmath_symbol_t symbol, pmath_symbol_attributes_t attr)

  Set a symbol's attributes.

- PMATH_API pmath_t pmath_symbol_get_value (pmath_symbol_t symbol)

  Get a symbol's value.

- PMATH_API void pmath_symbol_set_value (pmath_symbol_t symbol, pmath_t value)

  Set a symbol's value.

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

  Execute a function synchronized to a symbol.

- PMATH_API void pmath_symbol_update (pmath_symbol_t symbol)

  Update a symbol manually.

Related Functions

(Note that these are not member functions.)

- PMATH_API void pmath_collect_temporary_symbols (void)

  Collect unused symbols with the Temporary attribute.

8.17.1 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

8.17.2 Friends And Related Function Documentation

8.17.2.1 PMATH_API 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

8.18 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_API pmath_t pmath_ref(pmath_t obj)

  Increments the reference counter of an object and returns it.

- PMATH_API void pmath_unref(pmath_t obj)

  Decrements the reference counter of an object and frees its memory if the reference counter becomes 0.

- PMATH_API pmath_bool_t pmath_instance_of(pmath_t obj, pmath_type_t type)

  Determine whether an object has a specific type.

- PMATH_IS_MAGIC(obj)

  Fast test, whether an object is a ‘magic value‘.

- PMATH_API unsigned int pmath_hash (pmath_t obj)

  Calculates an object’s hash value.

- PMATH_API pmath_bool_t pmath_equals (pmath_t objA, pmath_t objB)

  Compares two objects for identity.

- PMATH_API int pmath_compare (pmath_t objA, pmath_t objB)

  Compares two objects syntactically.

- PMATH_API void pmath_write (pmath_t obj, pmath_write_options_t options, pmath_write_func_t write, void ∗user)

  Write an object to a stream.

- PMATH_API pmath_bool_t pmath_is_evaluated (pmath_t obj)

  Test whether an object is already evaluated.

Related Functions

(Note that these are not member functions.)

- PMATH_API pmath_t pmath_evaluate (pmath_t obj)

  Evaluate an object.

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

  Generate a List of objects with a format string.

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

  Generate a List of objects with a format string.

8.18.1 Detailed Description

The basic type of all pMath objects.

Note that this is not neccessaryly a pointer to some accessible piece of memory. Use pmath_instance_of() to determine whether an object is of a specific type or set of types.

You must free unused objects with pmath_unref().

8.18.2 Friends And Related Function Documentation

8.18.2.1 PMATH_API 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.

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

8.19 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 NULL if read_function is also 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.

8.19.1 Detailed Description

Access functions for text files.

See also:

pmath_file_create_text

The documentation for this class was generated from the following file:

- pmath-util/files.h

8.20 pmath_thread_t Class Reference

The Representation of a thread.

Related Functions

(Note that these are not member functions.)

- PMATH_API pmath_messages_t pmath_thread_fork_daemon (pmath_callback_t callback, pmath_callback_t kill, void ∗data)

  Create a new deamon thread.

- PMATH_API pmath_thread_t pmath_thread_get_current (void)

  Get the current pMath thread.

- PMATH_API pmath_thread_t pmath_thread_get_parent (pmath_thread_t thread)

  Get a thread's direct parent.

- PMATH_API 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_API 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.

8.20.1 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).

8.20.2 Friends And Related Function Documentation

8.20.2.1 PMATH_API 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 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_get_current()) will be allready 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.

The documentation for this class was generated from the following files:

- pmath-util/concurrency/threads.h
- pmath-util/concurrency/threadpool.h

8.21 pmath_threadlock_t Class Reference

A reentrant lock for threads.

Related Functions

(Note that these are not member functions.)

- PMATH_API void pmath_thread_call_locked (pmath_threadlock_t ∗threadlock_ptr, pmath_callback_t callback, void ∗data)

  Execute a function synchronized with a threadlock.

8.21.1 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

8.22 pmath::String Class Reference

A wrapper for pmath_string_t.

Inherits pmath::Expr.

Public Member Functions

- String (pmath_string_t _str=0) throw ()

  Construct form a pmath_string_t, stealing the reference.

- String (const char ∗latin1, int len=-1) throw ()

  Construct from Latin-1 encoded C string.

- String part (int start, int length) const throw ()

  Get string part.

- bool equals (const char ∗latin1) const throw ()

  Check for equality with a C string.

- bool starts_with (const String &s) const throw ()

  Check for prefix equality.

- bool starts_with (const char ∗s, int len=-1) const throw ()

  Check for prefix equality with a C string (Latin-1).

- bool starts_with (const uint16_t ∗s, int len=-1) const throw ()

Check for prefix equality with a UCS-2/UTF-16 string.

- 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 ()

  Insert a substring. Changes the object itself.

- void insert (int pos, const uint16_t ∗ucs2, int len=-1) throw ()

  Insert a substring. Changes the object itself.

- void remove (int start, int length) throw ()

  Remove a substring. Changes the object itself.

- 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 unicode character.

- static String FromUtf8 (const char ∗utf8, int len=-1) throw ()

  Construct from UTF-8 encoded C string.

8.22.1 Detailed Description

A wrapper for pmath_string_t.

This class provides some string utility functions in addition to Expr.

The documentation for this class was generated from the following file:

- pmath-cpp.h

Index