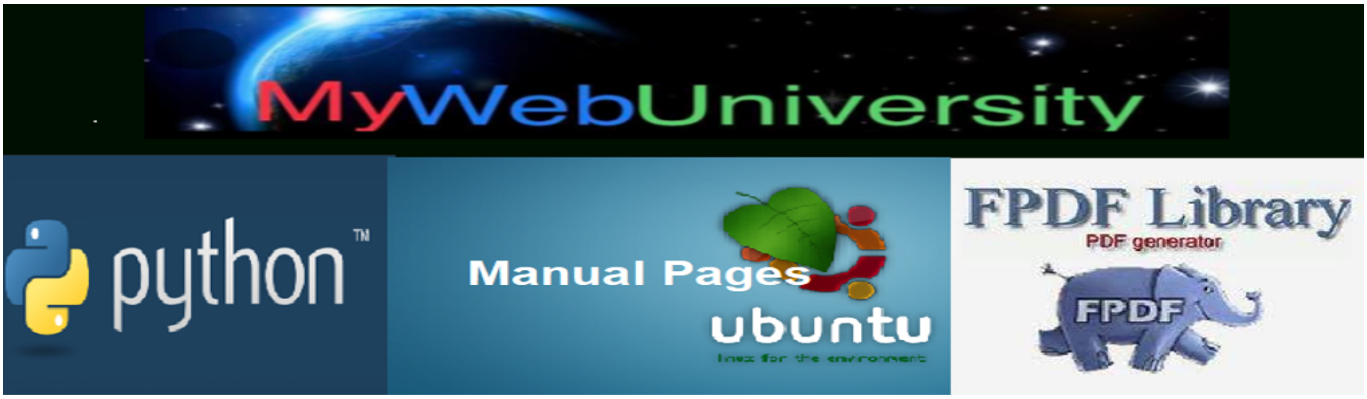

Rocky Enterprise Linux 9.2 Manual Pages on command 'Math::Int64.3pm'

$ man Math::Int64.3pm

Math::Int64(3pm) User Contributed Perl Documentation Math::Int64(3pm)

NAME

Math::Int64 - Manipulate 64 bits integers in Perl

SYNOPSIS

```
use Math::Int64 qw(int64 uint64);

my $i = int64(1);

my $j = $i << 40;

print($i + $j * 1000000);

my $k = uint64("12345678901234567890");
```

DESCRIPTION

This module adds support for 64 bit integers, signed and unsigned, to Perl.

Exportable functions

int64()

int64($value)

Creates a new int64 value and initializes it to $value, where $value can be a Perl number or a string containing a number.

For instance:

```
$i = int64(34);

$j = int64("-123454321234543212345");

$k = int64(123456769847848393898898 8); # wrong!!!
                                        #  the unquoted number would
                                        #  be converted first to a
                                        #  real number causing it to
```

```
                                # loose some precision.
```

Once the int64 number is created it can be manipulated as any other Perl value supporting all the standard operations (addition, negation, multiplication, postincrement, etc.).

net_to_int64($str)

Converts an 8 bytes string containing an int64 in network order to the internal representation used by this module.

int64_to_net($int64)

Returns an 8 bytes string with the representation of the int64 value in network order.

native_to_int64($str)

int64_to_native($int64)

similar to net_to_int64 and int64_to_net, but using the native CPU order.

int64_to_number($int64)

returns the optimum representation of the int64 value using Perl internal types (IV, UV or NV). Precision may be lost.

For instance:

```
  for my $l (10, 20, 30, 40, 50, 60) {
    my $i = int64(1) << $l;
    my $n = int64_to_number($i);
    print "int64:$i => perl:$n\n";
  }
```

string_to_int64($str, $base)

Converts the string to a int64 value. The conversion is done according to the given base, which must be a number between 2 and 36 inclusive or the special value 0. $base defaults to 0.

The string may begin with an arbitrary amount of white space followed by a single optional "+" or "-" sign. If base is zero or 16, the string may then include a "0x" prefix, and the number will be read in base 16; otherwise, a zero base is taken as 10 (decimal) unless the next character is '0', in which case it is taken as 8 (octal).

Underscore characters ("_") between the digits are ignored.

No overflow checks are performed by this function unless the "die_on_overflow" pragma is used (see "Die on overflow" below).

See also strtoll(3).

hex_to_int64($i64)

Shortcut for string_to_int64($str, 16)

int64_to_string($i64, $base)

Converts the int64 value to its string representation in the given base (defaults to 10).

int64_to_hex($i64)

Shortcut for "int64_to_string($i64, 16)".

int64_to_BER($i64)

Converts the int64 value to its BER representation (see "pack" in perlfunc for a description of the BER format).

In the case of signed numbers, they are transformed into unsigned numbers before encoding them in the BER format with the following rule:

```
$neg = ($i64 < 0 ? 1 : 0);
$u64 = (($neg ? ~$i64 : $i64) << 1) | $neg;
```

That way, positive and negative integers are interleaved as 0, -1, 1, 2, -2, ... The format is similar to that used by Google protocol buffers to encode signed variants but with the most significant groups first (protocol buffers uses the least significant groups first variant).

If you don't want that preprocessing for signed numbers, just use the "uint64_to_BER" function instead.

BER_to_int64($str)

Decodes the int64 number in BER format from the given string.

There must not be any extra bytes on the string after the encoded number.

BER_length($str)

Given a string with a BER encoded number at the beginning, this function returns the number of bytes it uses.

The right way to shift a BER encoded number from the beginning of some string is as follows:

```
$i64 = BER_to_int64(substr($str, 0, BER_length($str), ''));
```

int64_rand

Generates a 64 bit random number using ISAAC-64 algorithm.

int64_srand($seed)

int64_srand()

Sets the seed for the random number generator.

$seed, if given, should be a 2KB long string.

uint64

uint64_to_number

net_to_uint64

uint64_to_net

native_to_uint64

uint64_to_native

string_to_uint64

hex_to_uint64

uint64_to_string

uint64_to_hex

These functions are similar to their int64 counterparts, but manipulate 64 bit unsigned integers.

uint64_to_BER($u64)

Encodes the given unsigned integer in BER format (see "pack" in perlfunc).

BER_to_uint64($str)

Decodes from the given string an unsigned number in BER format.

uint64_rand

Generates a 64 bit random unsigned number using ISAAC-64 algorithm.

Die on overflow

The lexical pragma "Math::Int64::die_on_overflow" configures the module to throw an error when some operation results in integer overflow.

For instance:

```
use Math::Int64 qw(uint64);
use Math::Int64::die_on_overflow;
my $zero = uint64(0);
say ($zero - 1);                # dies as -1 falls outside
                                # the uint64_t range
no Math::Int64::die_on_overflow; # deactivates lexical pragma
say ($zero - 1);                # no error is detected here!
```

The pragma can also be activated as follows:

```
use Math::Int64 ':die_on_overflow';
```

Once this pragma is used, several Math::Int64 operations may become slower. Deactivating the pragma will not make them fast again.

On Perl 5.8.x, as lexical pragmas support is not available, the pragma "die_on_overflow" pragma is global and can not be deactivated.

Fallback to native 64bit support if available

If the lexical pragma "Math::Int64::native_if_available" is used in your program and the version of perl in use has native support for 64bit integers, the functions imported from the module that create 64bit integers (i.e. "uint64", "int64", "string_to_int64", "native_to_int64", etc.) will return regular perl scalars.

For instance:

```
use Math::Int64 qw(int64);
$a = int64(34); # always returns an object of the class Math::Int64
use Math::Int64::native_if_available;
$a = int64(34); # returns a regular scalar on perls compiled with
                # 64bit support
```

This feature is not enabled by default because the semantics for perl scalars and for 64 bit integers as implemented in this module are not identical.

Perl is prone to coerce integers into floats while this module keeps then always as 64bit integers. Specifically, the division operation and overflows are the most problematic cases. Also, when using native integers, the signed/unsigned division blurs.

Besides that, in most situations it is safe to use the native fallback.

As happens with the "die_on_overflow" pragma, on Perl 5.8.x it is global.

The pragma can also be activated as follows:

```
use Math::Int64 ':native_if_available';
```

Transparent conversion of objects to int64/uint64

When in some operation involving int64/uint64 numbers, a blessed object is passed as an operand, the module would try to coerce the object into an int64/uint64 number calling the methods "as_int64"/"as_uint64" respectively.

If the corresponding method is not implemented, the object will be stringified and then parsed as a base 10 number.

Storable integration

Objects of classes Math::Int64 and Math::UInt64 implement the STORABLE_freeze and STORABLE_thaw methods for a transparent integration with Storable.

C API

This module provides a native C API that can be used to create and read Math::Int64 int64 and uint64 SVs from your own XS modules.

In order to use it you need to follow these steps:

? Import the files "perl_math_int64.c", "perl_math_int64.h" and optionally "typemaps" from Math::Int64 "c_api_client" directory into your project directory.

? Include the file "perl_math_int64.h" in the C or XS source files where you want to convert 64bit integers to/from Perl SVs.

Note that this header file requires the types int64_t and uint64_t to be defined beforehand.

? Add the file "perl_math_int64.c" to your compilation targets (see the sample Makefile.PL below).

? Add a call to the macro "PERL_MATH_INT64_LOAD_OR_CROAK" into the "BOOT" section of your XS file.

For instance:

```
 --- Foo64.xs ---------
  #include "EXTERN.h"
  #include "perl.h"
  #include "XSUB.h"
  #include "ppport.h"
  /* #define MATH_INT64_NATIVE_IF_AVAILABLE */
  #include "math_int64.h"
  MODULE = Foo64               PACKAGE = Foo64
  BOOT:
     PERL_MATH_INT64_LOAD_OR_CROAK;
  int64_t
  some_int64()
  CODE:
     RETVAL = -42;
  OUTPUT:
     RETVAL
  --- Makefile.PL -----
  use ExtUtils::MakeMaker;
```

```
  WriteMakefile( NAME         => 'Foo64',
                 VERSION_FROM => 'lib/Foo64.pm',
                 OBJECT       => '$(O_FILES)' );
```

If the macro "MATH_INT64_NATIVE_IF_AVAILABLE" is defined before including "perl_math_int64.h" and the perl interpreter is compiled with native 64bit integer support, IVs will be used to represent 64bit integers instead of the object representation provided by Math::Int64.

These are the C macros available from Math::Int64 C API:

SV *newSVi64(int64_t i64)

Returns an SV representing the given int64_t value.

SV *newSVu64(uint64_t 64)

Returns an SV representing the given uint64_t value.

int64_t SvI64(SV *sv)

Extracts the int64_t value from the given SV.

uint64_t SvU64(SV *sv)

Extracts the uint64_t value from the given SV.

int SvI64OK(SV *sv)

Returns true is the given SV contains a valid int64_t value.

int SvU64OK(SV *sv)

Returns true is the given SV contains a valid uint64_t value.

uint64_t randU64(void)

Returns a random 64 bits unsigned integer.

SV sv_seti64(SV *sv, uint64_t i64)

Sets the value of the perl scalar to the given int64_t value.

SV sv_setu64(SV *sv, uint64_t i64)

Sets the value of the perl scalar to the given uint64_t value.

If you require any other function available through the C API don't hesitate to ask for it!

BUGS AND SUPPORT

The Storable integration feature is experimental.

The C API feature is experimental.

This module requires int64 support from the C compiler.

In order to report bugs you can send me and email to the address that appears below or use

the CPAN RT bug tracking system available at <http://rt.cpan.org>.

The source for the development version of the module is hosted at GitHub:

<https://github.com/salva/p5-Math-Int64>.

My wishlist

If you like this module and you're feeling generous, take a look at my Amazon Wish List:

<http://amzn.com/w/1WU1P6IR5QZ42>

SEE ALSO

The C API usage sample module Math::Int64::C_API::Sample.

Other modules providing support for larger integers or numbers are Math::BigInt, Math::BigRat and Math::Big, Math::BigInt::BitVect, Math::BigInt::Pari and Math::BigInt::GMP.

COPYRIGHT AND LICENSE