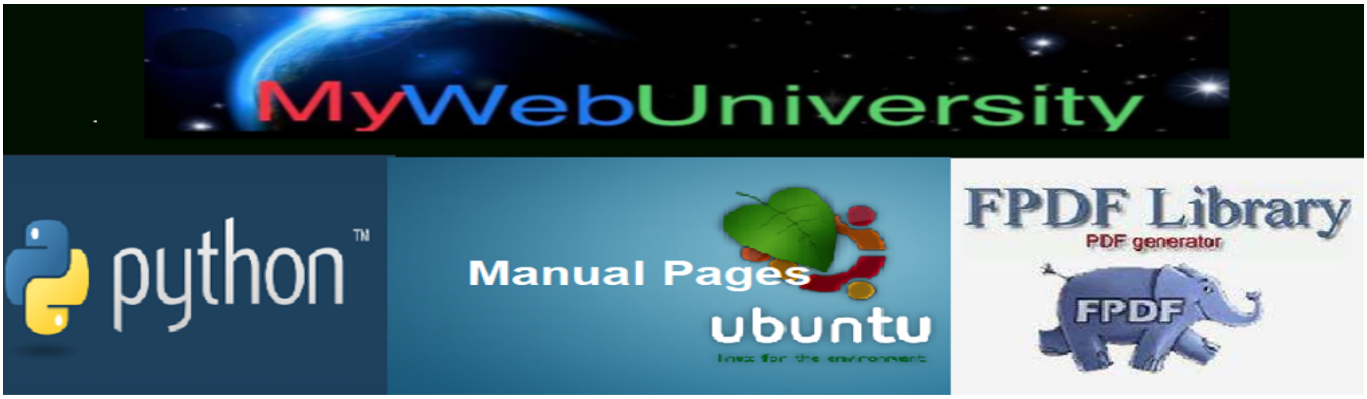

Rocky Enterprise Linux 9.2 Manual Pages on command 'Math::BigInt::Lib.3perl'

$ man Math::BigInt::Lib.3perl

NAME

Math::BigInt::Lib - virtual parent class for Math::BigInt libraries

SYNOPSIS

```
# In the backend library for Math::BigInt et al.
package Math::BigInt::MyBackend;
use Math::BigInt::lib;
our @ISA = qw< Math::BigInt::lib >;
sub _new { ... }
sub _str { ... }
sub _add { ... }
str _sub { ... }
...
# In your main program.
use Math::BigInt lib => 'MyBackend';
```

DESCRIPTION

This module provides support for big integer calculations. It is not intended to be used directly, but rather as a parent class for backend libraries used by Math::BigInt, Math::BigFloat, Math::BigRat, and related modules.

Other backend libraries include Math::BigInt::Calc, Math::BigInt::FastCalc, Math::BigInt::GMP, and Math::BigInt::Pari.

In order to allow for multiple big integer libraries, Math::BigInt was rewritten to use a plug-in library for core math routines. Any module which conforms to the API can be used

by Math::BigInt by using this in your program:

```
use Math::BigInt lib => 'libname';
```

'libname' is either the long name, like 'Math::BigInt::Pari', or only the short version, like 'Pari'.

General Notes

A library only needs to deal with unsigned big integers. Testing of input parameter validity is done by the caller, so there is no need to worry about underflow (e.g., in "_sub()" and "_dec()") or about division by zero (e.g., in "_div()" and "_mod()")) or similar cases.

Some libraries use methods that don't modify their argument, and some libraries don't even use objects, but rather unblessed references. Because of this, liberary methods are always called as class methods, not instance methods:

```
$x = Class -> method($x, $y);     # like this
$x = $x -> method($y);            # not like this ...
$x -> method($y);                 # ... or like this
```

And with boolean methods

```
$bool = Class -> method($x, $y);  # like this
$bool = $x -> method($y);         # not like this
```

Return values are always objects, strings, Perl scalars, or true/false for comparison routines.

API version

CLASS->api_version()

This method is no longer used and can be omitted. Methods that are not implemented by a subclass will be inherited from this class.

Constructors

The following methods are mandatory: _new(), _str(), _add(), and _sub(). However, computations will be very slow without _mul() and _div().

CLASS->_new(STR)

Convert a string representing an unsigned decimal number to an object representing the same number. The input is normalized, i.e., it matches "^(0|[1-9]\d*)$".

CLASS->_zero()

Return an object representing the number zero.

CLASS->_one()

Return an object representing the number one.

CLASS->_two()

Return an object representing the number two.

CLASS->_ten()

Return an object representing the number ten.

CLASS->_from_bin(STR)

Return an object given a string representing a binary number. The input has a '0b' prefix and matches the regular expression "^0[bB](0|1[01]*)$".

CLASS->_from_oct(STR)

Return an object given a string representing an octal number. The input has a '0' prefix and matches the regular expression "^0[1-7]*$".

CLASS->_from_hex(STR)

Return an object given a string representing a hexadecimal number. The input has a '0x' prefix and matches the regular expression "^0x(0|[1-9a-fA-F][\da-fA-F]*)$".

CLASS->_from_bytes(STR)

Returns an object given a byte string representing the number. The byte string is in big endian byte order, so the two-byte input string "\x01\x00" should give an output value representing the number 256.

CLASS->_from_base(STR, BASE, COLLSEQ)

Returns an object given a string STR, a base BASE, and a collation sequence COLLSEQ. Each character in STR represents a numerical value identical to the character's position in COLLSEQ. All characters in STR must be present in COLLSEQ.

If BASE is less than or equal to 94, and a collation sequence is not specified, the following default collation sequence is used. It contains of all the 94 printable ASCII characters except space/blank:

```
0123456789                # ASCII  48 to  57
ABCDEFGHIJKLMNOPQRSTUVWXYZ  # ASCII  65 to  90
abcdefghijklmnopqrstuvwxyz  # ASCII  97 to 122
!"#$%&'()*+,-./           # ASCII  33 to  47
:;<=>?@                   # ASCII  58 to  64
[\]^_`                    # ASCII  91 to  96
{|}~                      # ASCII 123 to 126
```

If the default collation sequence is used, and the BASE is less than or equal to 36,

the letter case in STR is ignored.

For instance, with base 3 and collation sequence "-/|", the character "-" represents 0, "/" represents 1, and "|" represents 2. So if STR is "/|-", the output is 1 * 3**2 + 2 * 3**1 + 0 * 3**0 = 15.

The following examples show standard binary, octal, decimal, and hexadecimal conversion. All examples return 250.

```
$x = $class -> _from_base("11111010", 2)
$x = $class -> _from_base("372", 8)
$x = $class -> _from_base("250", 10)
$x = $class -> _from_base("FA", 16)
```

Some more examples, all returning 250:

```
$x = $class -> _from_base("100021", 3)
$x = $class -> _from_base("3322", 4)
$x = $class -> _from_base("2000", 5)
$x = $class -> _from_base("caaa", 5, "abcde")
$x = $class -> _from_base("42", 62)
$x = $class -> _from_base("2!", 94)
```

Mathematical functions

CLASS->_add(OBJ1, OBJ2)

Returns the result of adding OBJ2 to OBJ1.

CLASS->_mul(OBJ1, OBJ2)

Returns the result of multiplying OBJ2 and OBJ1.

CLASS->_div(OBJ1, OBJ2)

In scalar context, returns the quotient after dividing OBJ1 by OBJ2 and truncating the result to an integer. In list context, return the quotient and the remainder.

CLASS->_sub(OBJ1, OBJ2, FLAG)

CLASS->_sub(OBJ1, OBJ2)

Returns the result of subtracting OBJ2 by OBJ1. If "flag" is false or omitted, OBJ1 might be modified. If "flag" is true, OBJ2 might be modified.

CLASS->_dec(OBJ)

Returns the result after decrementing OBJ by one.

CLASS->_inc(OBJ)

Returns the result after incrementing OBJ by one.

CLASS->_mod(OBJ1, OBJ2)

Returns OBJ1 modulo OBJ2, i.e., the remainder after dividing OBJ1 by OBJ2.

CLASS->_sqrt(OBJ)

Returns the square root of OBJ, truncated to an integer.

CLASS->_root(OBJ, N)

Returns the Nth root of OBJ, truncated to an integer.

CLASS->_fac(OBJ)

Returns the factorial of OBJ, i.e., the product of all positive integers up to and including OBJ.

CLASS->_dfac(OBJ)

Returns the double factorial of OBJ. If OBJ is an even integer, returns the product of all positive, even integers up to and including OBJ, i.e., 2*4*6*...*OBJ. If OBJ is an odd integer, returns the product of all positive, odd integers, i.e., 1*3*5*...*OBJ.

CLASS->_pow(OBJ1, OBJ2)

Returns OBJ1 raised to the power of OBJ2. By convention, 0**0 = 1.

CLASS->_modinv(OBJ1, OBJ2)

Returns the modular multiplicative inverse, i.e., return OBJ3 so that

(OBJ3 * OBJ1) % OBJ2 = 1 % OBJ2

The result is returned as two arguments. If the modular multiplicative inverse does not exist, both arguments are undefined. Otherwise, the arguments are a number (object) and its sign ("+" or "-").

The output value, with its sign, must either be a positive value in the range 1,2,...,OBJ2-1 or the same value subtracted OBJ2. For instance, if the input arguments are objects representing the numbers 7 and 5, the method must either return an object representing the number 3 and a "+" sign, since (3*7) % 5 = 1 % 5, or an object representing the number 2 and a "-" sign, since (-2*7) % 5 = 1 % 5.

CLASS->_modpow(OBJ1, OBJ2, OBJ3)

Returns the modular exponentiation, i.e., (OBJ1 ** OBJ2) % OBJ3.

CLASS->_rsft(OBJ, N, B)

Returns the result after shifting OBJ N digits to thee right in base B. This is equivalent to performing integer division by B**N and discarding the remainder, except that it might be much faster.

For instance, if the object $obj represents the hexadecimal number 0xabcde, then

"_rsft($obj, 2, 16)" returns an object representing the number 0xabc. The "remainer", 0xde, is discarded and not returned.

CLASS->_lsft(OBJ, N, B)

Returns the result after shifting OBJ N digits to the left in base B. This is equivalent to multiplying by B**N, except that it might be much faster.

CLASS->_log_int(OBJ, B)

Returns the logarithm of OBJ to base BASE truncted to an integer. This method has two output arguments, the OBJECT and a STATUS. The STATUS is Perl scalar; it is 1 if OBJ is the exact result, 0 if the result was truncted to give OBJ, and undef if it is unknown whether OBJ is the exact result.

CLASS->_gcd(OBJ1, OBJ2)

Returns the greatest common divisor of OBJ1 and OBJ2.

CLASS->_lcm(OBJ1, OBJ2)

Return the least common multiple of OBJ1 and OBJ2.

CLASS->_fib(OBJ)

In scalar context, returns the nth Fibonacci number: _fib(0) returns 0, _fib(1) returns 1, _fib(2) returns 1, _fib(3) returns 2 etc. In list context, returns the Fibonacci numbers from F(0) to F(n): 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

CLASS->_lucas(OBJ)

In scalar context, returns the nth Lucas number: _lucas(0) returns 2, _lucas(1) returns 1, _lucas(2) returns 3, etc. In list context, returns the Lucas numbers from L(0) to L(n): 2, 1, 3, 4, 7, 11, 18, 29,47, 76, ...

Bitwise operators

CLASS->_and(OBJ1, OBJ2)

Returns bitwise and.

CLASS->_or(OBJ1, OBJ2)

Returns bitwise or.

CLASS->_xor(OBJ1, OBJ2)

Returns bitwise exclusive or.

CLASS->_sand(OBJ1, OBJ2, SIGN1, SIGN2)

Returns bitwise signed and.

CLASS->_sor(OBJ1, OBJ2, SIGN1, SIGN2)

Returns bitwise signed or.

CLASS->_sxor(OBJ1, OBJ2, SIGN1, SIGN2)

Returns bitwise signed exclusive or.

Boolean operators

CLASS->_is_zero(OBJ)

Returns a true value if OBJ is zero, and false value otherwise.

CLASS->_is_one(OBJ)

Returns a true value if OBJ is one, and false value otherwise.

CLASS->_is_two(OBJ)

Returns a true value if OBJ is two, and false value otherwise.

CLASS->_is_ten(OBJ)

Returns a true value if OBJ is ten, and false value otherwise.

CLASS->_is_even(OBJ)

Return a true value if OBJ is an even integer, and a false value otherwise.

CLASS->_is_odd(OBJ)

Return a true value if OBJ is an even integer, and a false value otherwise.

CLASS->_acmp(OBJ1, OBJ2)

Compare OBJ1 and OBJ2 and return -1, 0, or 1, if OBJ1 is numerically less than, equal to, or larger than OBJ2, respectively.

String conversion

CLASS->_str(OBJ)

Returns a string representing OBJ in decimal notation. The returned string should have no leading zeros, i.e., it should match "^(0|[1-9]\d*)$".

CLASS->_to_bin(OBJ)

Returns the binary string representation of OBJ.

CLASS->_to_oct(OBJ)

Returns the octal string representation of the number.

CLASS->_to_hex(OBJ)

Returns the hexadecimal string representation of the number.

CLASS->_to_bytes(OBJ)

Returns a byte string representation of OBJ. The byte string is in big endian byte order, so if OBJ represents the number 256, the output should be the two-byte string "\x01\x00".

CLASS->_to_base(OBJ, BASE, COLLSEQ)

Returns a string representation of OBJ in base BASE with collation sequence COLLSEQ.

```
$val = $class -> _new("210");
$str = $class -> _to_base($val, 10, "xyz")  # $str is "zyx"
$val = $class -> _new("32");
$str = $class -> _to_base($val, 2, "-|")  # $str is "|-----"
```

See _from_base() for more information.

CLASS->_as_bin(OBJ)

Like "_to_bin()" but with a '0b' prefix.

CLASS->_as_oct(OBJ)

Like "_to_oct()" but with a '0' prefix.

CLASS->_as_hex(OBJ)

Like "_to_hex()" but with a '0x' prefix.

CLASS->_as_bytes(OBJ)

This is an alias to "_to_bytes()".

Numeric conversion

CLASS->_num(OBJ)

Returns a Perl scalar number representing the number OBJ as close as possible. Since Perl scalars have limited precision, the returned value might not be exactly the same as OBJ.

Miscellaneous

CLASS->_copy(OBJ)

Returns a true copy OBJ.

CLASS->_len(OBJ)

Returns the number of the decimal digits in OBJ. The output is a Perl scalar.

CLASS->_zeros(OBJ)

Returns the number of trailing decimal zeros. The output is a Perl scalar. The number zero has no trailing decimal zeros.

CLASS->_digit(OBJ, N)

Returns the Nth digit in OBJ as a Perl scalar. N is a Perl scalar, where zero refers to the rightmost (least significant) digit, and negative values count from the left (most significant digit). If $obj represents the number 123, then

```
CLASS->_digit($obj,  0)     # returns 3
CLASS->_digit($obj,  1)     # returns 2
```

```
    CLASS->_digit($obj,  2)     # returns 1
    CLASS->_digit($obj, -1)     # returns 1
```

CLASS->_digitsum(OBJ)

Returns the sum of the base 10 digits.

CLASS->_check(OBJ)

Returns true if the object is invalid and false otherwise. Preferably, the true value is a string describing the problem with the object. This is a check routine to test the internal state of the object for corruption.

CLASS->_set(OBJ)

xxx

API version 2

The following methods are required for an API version of 2 or greater.

Constructors

CLASS->_1ex(N)

Return an object representing the number 10**N where N >= 0 is a Perl scalar.

Mathematical functions

CLASS->_nok(OBJ1, OBJ2)

Return the binomial coefficient OBJ1 over OBJ1.

Miscellaneous

CLASS->_alen(OBJ)

Return the approximate number of decimal digits of the object. The output is a Perl scalar.

WRAP YOUR OWN

If you want to port your own favourite C library for big numbers to the Math::BigInt interface, you can take any of the already existing modules as a rough guideline. You should really wrap up the latest Math::BigInt and Math::BigFloat testsuites with your module, and replace in them any of the following:

```
    use Math::BigInt;
```

by this:

```
    use Math::BigInt lib => 'yourlib';
```

This way you ensure that your library really works 100% within Math::BigInt.

BUGS

Please report any bugs or feature requests to "bug-math-bigint at rt.cpan.org", or through

the web interface at <https://rt.cpan.org/Ticket/Create.html?Queue=Math-BigInt> (requires login). We will be notified, and then you'll automatically be notified of progress on your bug as I make changes.

SUPPORT

You can find documentation for this module with the perldoc command.

```
perldoc Math::BigInt::Calc
```

You can also look for information at:

- RT: CPAN's request tracker
  <https://rt.cpan.org/Public/Dist/Display.html?Name=Math-BigInt>
- AnnoCPAN: Annotated CPAN documentation
  <http://annocpan.org/dist/Math-BigInt>
- CPAN Ratings
  <https://cpanratings.perl.org/dist/Math-BigInt>
- MetaCPAN
  <https://metacpan.org/release/Math-BigInt>
- CPAN Testers Matrix
  <http://matrix.cpantesters.org/?dist=Math-BigInt>
- The Bignum mailing list
  - Post to mailing list
    "bignum at lists.scsys.co.uk"
  - View mailing list
    <http://lists.scsys.co.uk/pipermail/bignum/>
  - Subscribe/Unsubscribe
    <http://lists.scsys.co.uk/cgi-bin/mailman/listinfo/bignum>

LICENSE

This program is free software; you may redistribute it and/or modify it under the same terms as Perl itself.

AUTHOR

Peter John Acklam, <pjacklam@online.no>

Code and documentation based on the Math::BigInt::Calc module by Tels <nospam-abuse@bloodgate.com>

SEE ALSO

Math::BigInt, Math::BigInt::Calc, Math::BigInt::GMP, Math::BigInt::FastCalc and

Math::BigInt::Pari.