

# Erlang BLAS 1.1.0 Documentation

August 2023



# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Reference</b>	<b>4</b>
2.1	run . . . . .	4
2.1.1	run/1 . . . . .	4
2.1.2	run/2 . . . . .	4
2.2	new . . . . .	5
2.2.1	new/1 . . . . .	5
2.2.2	new/2 . . . . .	5
2.3	shift . . . . .	6
2.3.1	shift/2 . . . . .	6
2.4	copy . . . . .	6
2.4.1	copy/2 . . . . .	6
2.5	to_bin . . . . .	6
2.5.1	to_bin/1 . . . . .	6
2.5.2	to_bin/2 . . . . .	6
2.6	to_list . . . . .	7
2.6.1	to_list/2 . . . . .	7
<b>3</b>	<b>Representing BLAS-LAPACKE types in Erlang</b>	<b>8</b>
3.1	Numbers - Arrays - Characters . . . . .	8
3.2	Enumerations . . . . .	8
<b>4</b>	<b>Examples</b>	<b>9</b>
4.1	dscal . . . . .	9
4.2	dgemm . . . . .	9
4.3	stpmv . . . . .	9
<b>5</b>	<b>Supported BLAS functions</b>	<b>10</b>
5.1	By default . . . . .	10
5.2	GRISP . . . . .	12

# 1 Introduction

This project, funded by the Erlang Ecosystem Foundation (<https://erlef.org/>), was made possible thanks to Peerst Stritzinger (<https://www.stritzinger.com/>) and aims to bring the efficiency of the BLAS-LAPACKE library to Erlang.

This document is a reference for the Erlang BLAS wrapper, and is not a reference of BLAS-LAPACKE. Such references can be found at:

- netlib: <https://netlib.org/>
- IBM: <https://www.ibm.com/docs/en>
- intel: <https://www.intel.com/content/www/us/en/content-details/671183/developer-reference-for-intel-math-kernel-library-intel-mkl-11-3-c.html?wapkw=BLAS%20mlk>

Thanks to the help of Luca Succi and Peerst Stritzinger, it is supported on the GRISP embedded device (<https://www.grisp.org/>).

Finally, the project maintainer can be contacted at [losseautanguy@gmail.com](mailto:losseautanguy@gmail.com).

## 2 Reference

The BLAS library exports the following functions:

```
[run/1, run/2, new/1, new/2, shift/2, copy/2, to_bin/1,  
to_bin/2, to_list/2, predictor/0]
```

The following record will be later referenced as `c_binary`:

```
-record(c_binary, {size, offset, resource}).
```

Finally, possible values of `blas_name` are provided in section "Supported BLAS functions".

### 2.1 run

#### 2.1.1 run/1

```
run(Tuple)  
  Tuple: {blas_name, Arg0, ..., ArgN}
```

Same as `run/2`. The first time it is executed, runs a benchmark of `dgemm` (see `predictor`); BLAS execution duration is then predicted and forwarded to `run/2`. Due to their diversity/complexity, LAPACKE functions are always sent to a dirty scheduler for one millisecond.

#### 2.1.2 run/2

```
run(Tuple, Scheduling)  
  Tuple: {blas\_name, Arg0, ..., ArgN}  
  Scheduling: Integer | dirty | clean
```

Tuple groups the name of requested BLAS functions, and its arguments. The section "Representing BLAS-LAPACKE types in Erlang" describes how to construct the latter. Scheduling is either:

- `dirty`: schedule for 1.5 ms on a dirty scheduler.
- `clean`: schedule for 0.5 ms on a clean scheduler.
- `Integer`: the percentage of 1ms expected to be used. If inferior to 100, clean scheduling will be used; otherwise dirty scheduling will be used.

If the `blas_name` executed without error, returns `ok`. It might raise the following exceptions:

- "Unknown blas." if the `blas_name` is not recognised.

- "Array overflow." if one of the inputs arrays is too small. Currently, only BLAS functions check for arrays overflow.
- "Invalid number of arguments." if too many arguments are present in the Tuple.
- "Could not translate argument I." if ArgI of Tuple could not be read.

## 2.2 new

This function is used to create a `c_binary`. The BLAS library executes in place and require mutable arrays. Though it could possible to do this with erlang binaries, it is safer not to; instead, a nif resource is used and stored in a `c_binary`.

### 2.2.1 new/1

```
new(Type)
  Type: Integer | Binary
```

If Type is Integer, allocates a `c_binary` of given byte size.  
 If Type is Binary, copy the input Binary into a `c_binary`.  
 Returns a `c_binary`.

### 2.2.2 new/2

```
new(Encoding, List)
  Encoding: int32 | int64 | s | float32 | d | float64
           | c | complex64 | z | complex128
```

List is a list of numbers to write in a new `c_binary`.  
 Encoding indicates how the numbers should be encoded:

- int32: integers of 32 bits.
- int64: integers of 64 bits.
- s, float32: floats of 32 bits.
- d, float64: floats of 64 bits.
- c, complex64: pair number of float of 32 bits.
- z, complex128: pair number of floats of 64 bits.

Returns a `c_binary`.

## 2.3 shift

The BLAS library tended to use interleaved matrices and arrays. In order to access them, shifting pointers/c\_binaries around is required.

### 2.3.1 shift/2

```
shift(Shift, C_binary)
    Shift: integer
    C_binary: c_binary
```

Returns a c\_binary witch starts with offset Shift (in bytes) relative to input C\_binary.

## 2.4 copy

This functions copies the content of a Binary into a c\_binary.

### 2.4.1 copy/2

```
copy(Binary, C_binary)
    Binary: binary
    C_binary: c_binary
```

Returns ok on success.

## 2.5 to\_bin

This functions converts a c\_binary to a binary.

### 2.5.1 to\_bin/1

```
to_bin(C_binary)
    C_binary: c_binary
```

Returns and Erlang binary copy of the c\_binary content.

### 2.5.2 to\_bin/2

```
to_bin(Size, C_binary)
    Size: integer
    C_binary: c_binary
```

Returns the first Size bytes of C\_binary copied in a binary.

## 2.6 to\_list

This function converts a Binary to a list with given encoding.

### 2.6.1 to\_list/2

```
to_list(Encoding, Binary)
  Encoding: int32 | int64 | s | float32 | d | float64
           | c | complex64 | z | complex128
  C_binary: c_binary
```

Encoding indicates how the numbers should be encoded:

- int32: integers of 32 bits.
- int64: integers of 32 bits.
- s, float32: floats of 32 bits.
- d, float64: floats of 64 bits.
- c, complex64: pair number of float of 32 bits.
- z, complex128: pair number of floats of 64 bits.

Returns a list of numbers contained by the c\_binary.

### 3 Representing BLAS-LAPACKE types in Erlang

This projects provide a complete interface to all BLAS-LAPACKE variables-types.

#### 3.1 Numbers - Arrays - Characters

```
char:          atom
const int:     int,
const float:   double,
const double:  double,
const int*:    binary, c_binary,
const float*:  binary, c_binary,
const double*: binary, c_binary,
const void*:   binary, c_binary,
const int*:    c_binary,
void*:         c_binary,
float*:        c_binary,
double*:       c_binary,
```

#### 3.2 Enumerations

Enumeration values are represented as atoms.

```
CBLAS_ORDER      blasRowMajor, blasColMajor
CBLAS_TRANSPOSE   n, blasNoTrans, t, blasTrans, c, blasConjTrans
CBLAS_UPLO        u, blasUpper, l, blasLower
CBLAS_DIAG        n, blasNonUnit, u, blasUnit
CBLAS_SIDE        l, blasLeft, r, blasRight
```



## 4 Examples

### 4.1 dscal

Double type, SCALE a vector.

X:  $\text{Alpha} * X$ .

cblas signature:

```
void cblas_dscal (const int n, const double a, double *x, const int incx);
```

Erlang code:

```
X = blas:new(float64, [1,2,1,2,1,2,1,2]),
ok = blas:run({dscal, 8, 2.0, X, 1}),
blas:bt1(float64, blas:to_bin(X)).
```

### 4.2 dgemm

Double type, GEneral matrices, Matrix Matrix product.

C:  $\text{Alpha} * A * B + \text{Beta} * C$

cblas signature:

```
void cblas_dgemm(CBLAS_LAYOUT layout, CBLAS_TRANSPOSE TransA,
                 CBLAS_TRANSPOSE TransB, const CBLAS_INT M, const CBLAS_INT N,
                 const CBLAS_INT K, const double alpha, const double *A,
                 const CBLAS_INT lda, const double *B, const CBLAS_INT ldb,
                 const double beta, double *C, const CBLAS_INT ldc);
)
```

Erlang code:

```
A = blas:new(float64, [1,2,3, 1,2,3, 1,2,3]),
B = blas:new(float64, [4,5,6, 4,5,6, 4,5,6]),
C = blas:new(float64, [0,0,0, 0,0,0, 0,0,0]),
ok = blas:run({dgemm, blasRowMajor, n,n, 3,3,3, 1.0, A,3, B,3, 0.0, C,3}).
```

### 4.3 stpmv

Single real numbers, Triangular Packed matrix, Matrix\*Vector operation.

X:  $A * X$ .

cblas signature:

```
void cblas_stpmv(CBLAS_LAYOUT layout, CBLAS_UPLO Uplo,
                 CBLAS_TRANSPOSE TransA, CBLAS_DIAG Diag,
                 const CBLAS_INT N, const float *Ap, float *X, const CBLAS_INT incX);
```

Erlang code:

```

N = 3,
A = blas:new(s, [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]),
X = blas:new(s, [-0.25,-0.125,0.5]),
ok = blas:run(
    {stpmv, blasRowMajor, blasUpper, blasNoTrans, blasNonUnit, N, A, X, 1},
    clean
),
[1.0,2.0,3.0] = blas:to_list(s, X).

```

## 5 Supported BLAS functions

### 5.1 By default

The following blas.atom are supported:

saxpy, daxpy, caxpy, zaxpy, scopy, dcopy, ccopy, zcopy, sswap, dswap, cswap, zswap, sscal, dscal, cscal, csscal, zscal, zdscal, sdot, ddot, cdotu, zdotu, cdotc, zdotc, dsdot, sdsdot, snrm2, dnrm2, scnrm2, dznrm2, sasum, dasum, scasum, dzasum, isamax, idamax, icamax, izamax, srot, drot, csrot, zdrot, srotg, drotg, crotg, zrotg, srotmg, drotmg, srotm, drotm, isamin, idamin, icamin, izamin, ismax, idmax, icmax, izmax, ismin, idmin, icmin, izmin, sgemv, dgemv, cgemv, zgemv, sgbm, dgbmv, cgbmv, zgbmv, ssbmv, dsbmv, sger, dger, strmv, dtrmv, ctrmv, ztrmv, strsv, dtrsv, ctrsv, ztrsv, strsm, dtrsm, ctrsm, ztrsm, cgeru, cgerc, zgeru, zgerc, sgemm, dgemm, cgemm, zgemm, cgemm3m, zgemm, zgemm3m, stbmv, dtbmv, ctbmv, ztbmv, stbsv, dtbsv, ctbsv, ztbsv, stpmv, dtpmv, ctpmv, ztpmv, stpsv, dtpsv, ctpsv, ztpsv, ssymv, dsymv, chemv, zhemv, sspmv, dspmv, sspr, dspr, chpr, zhpr, sspr2, dspr2, chpr2, zhpr2, chbm, zhbmv, chpmv, zhpmv, cher, zher, chemm, zhemm, cherk, zherk, cher2k, zher2k, ssymm, dsymm, csymm, zsymm, ssyrk, dsyrk, csyrk, zsyrk, ssyr2k, dsyr2k, csyr2k, zsyr2k, ssum, dsum, dzsum, scsum, cher2, zher2, strmm, dtrmm, ctrmm, ztrmm, ssyr, dsyr, ssyr2, dsyr2, sbdsdc, dbdsdc, sbdsqr, dbdsqr, cbdsqr, zbdsqr, sdisna, ddisna, sgbb, dgb, cgb, zgb, sgbbcon, dgbcon, cgbcon, zgbcon, sgbe, dgbe, cgbe, zgbe, sgbequb, dgbequb, cgbequb, zgbqub, sgbrfs, dgbrfs, cgbrfs, zgbrfs, sgbsv, dgbsv, cgbsv, zgbsv, sgbrf, dgbrf, cgbrf, zgbrf, sgbrs, dgbrs, cgbrs, zgbrs, sgebal, dgebal, cgebal, zgebal, sgebrd, dgebrd, cgebrd, zgebrd, sgecon, dgecon, cgecon, zgecon, sgeequ, dgeequ, cgeequ, zgeequ, sgeequb, dgeequb, cgeequb, zgeequb, sgeev, dgeev, cgeev, zgeev, sgeevx, dgeevx, cgeevx, zgeevx, sgehrd, dgehrd, cgehrd, zgehrd, sgejsv, dgejsv, sgelqf, dgelqf, cgelqf, zgelqf, sgels, dgels, cgels, zgels, sgelsd, dgelsd, cgelsd, zgelsd, sgels, dgels, cgels, zgels, sgelsy, dgelsy, cgelsy, zgelsy, sgeqlf, dgeqlf, cgelqf, zgelqf, sgeqp3, dgeqp3, cgeqp3, zgeqp3, sgeqpf, dgeqpf, cgeqpf, zgeqpf, sgeqrf, dgeqrf, cgeqrf, zgeqrf, sgeqrfp, dgeqrfp, cgeqrfp, zgeqrfp, sgerfs, dgerfs, cgerfs, zgerfs, sgerqf, dgerqf, cgerqf, zgerqf, sgesdd, dgesdd, cgesdd, zgesdd, sgesv, dgesv, cgesv, zgesv, sgesvd, dgesvd, cgesvd, zgesvd, sgesvj, dgesvj, cgesvj, zgesvj, sgetrf, dgetrf, cgetrf, zgetrf, sgetri, dgetri



## 5.2 GRISP

12