ReproBLAS

Generated by Doxygen 1.8.10

Mon Jan 18 2016 14:14:50

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 include/idxd.h File Reference

idxd.h defines the indexed types and the lower level functions associated with their use.

```
#include <stddef.h>
#include <stdlib.h>
#include <float.h>
```

Macros

- #define DIWIDTH 40

  Indexed double precision bin width.
- #define SIWIDTH 13

  Indexed single precision bin width.
- #define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)

  Indexed double precision maximum index.
- #define idxd_SIMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SIWIDTH) - 1)

  Indexed single precision maximum index.
- #define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)

  The maximum double precision fold supported by the library.
- #define idxd_SIMAXFOLD (idxd_SIMAXINDEX + 1)

  The maximum single precision fold supported by the library.
- #define idxd_DIENDURANCE (1 << (DBL_MANT_DIG - DIWIDTH - 2))

  Indexed double precision deposit endurance.
- #define idxd_SIENDURANCE (1 << (FLT_MANT_DIG - SIWIDTH - 2))

  Indexed single precision deposit endurance.
- #define idxd_DICAPACITY (idxd_DIENDURANCE∗(1.0/DBL_EPSILON - 1.0))

  Indexed double precision capacity.
- #define idxd_SICAPACITY (idxd_SIENDURANCE∗(1.0/FLT_EPSILON - 1.0))

  Indexed single precision capacity.
- #define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

  Indexed double precision compression factor.
- #define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

  Indexed single precision compression factor.
- #define idxd_DMEXPANSION (1.0∗(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

  Indexed double precision expansion factor.
- #define idxd_SMEXPANSION (1.0∗(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

  Indexed single precision expansion factor.

Typedefs

- typedef double double_indexed

  The indexed double datatype.
- typedef double double_complex_indexed

  The indexed complex double datatype.
- typedef float float_indexed

  The indexed float datatype.
- typedef float float_complex_indexed

  The indexed complex float datatype.

Functions

- size_t **idxd_disize** (const int fold)
- size_t **idxd_zisize** (const int fold)
- size_t **idxd_sisize** (const int fold)
- size_t **idxd_cisize** (const int fold)
- double_indexed ∗ **idxd_dialloc** (const int fold)
- double_complex_indexed ∗ **idxd_zialloc** (const int fold)
- float_indexed ∗ **idxd_sialloc** (const int fold)
- float_complex_indexed ∗ **idxd_cialloc** (const int fold)
- int **idxd_dinum** (const int fold)
- int **idxd_zinum** (const int fold)
- int **idxd_sinum** (const int fold)
- int **idxd_cinum** (const int fold)
- double **idxd_dibound** (const int fold, const int N, const double X, const double S)
- float **idxd_sibound** (const int fold, const int N, const float X, const float S)
- const double ∗ **idxd_dmbins** (const int X)
- const float ∗ **idxd_smbins** (const int X)
- int **idxd_dindex** (const double X)
- int **idxd_dmindex** (const double ∗priX)
- int **idxd_dmindex0** (const double ∗priX)
- int **idxd_sindex** (const float X)
- int **idxd_smindex** (const float ∗priX)
- int **idxd_smindex0** (const float ∗priX)
- int **idxd_dmdenorm** (const int fold, const double ∗priX)
- int **idxd_zmdenorm** (const int fold, const double ∗priX)
- int **idxd_smdenorm** (const int fold, const float ∗priX)
- int **idxd_cmdenorm** (const int fold, const float ∗priX)
- void **idxd_diprint** (const int fold, const double_indexed ∗X)
- void **idxd_dmprint** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX)
- void **idxd_ziprint** (const int fold, const double_complex_indexed ∗X)
- void **idxd_zmprint** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX)
- void **idxd_siprint** (const int fold, const float_indexed ∗X)
- void **idxd_smprint** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX)
- void **idxd_ciprint** (const int fold, const float_complex_indexed ∗X)
- void **idxd_cmprint** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX)
- void **idxd_didiset** (const int fold, const double_indexed ∗X, double_indexed ∗Y)
- void **idxd_dmdmset** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_ziziset** (const int fold, const double_complex_indexed ∗X, double_complex_indexed ∗Y)
- void **idxd_zmzmset** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_zidiset** (const int fold, const double_indexed ∗X, double_complex_indexed ∗Y)

- void **idxd_zmdmset** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_sisiset** (const int fold, const float_indexed ∗X, float_indexed ∗Y)
- void **idxd_smsmset** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_ciciset** (const int fold, const float_complex_indexed ∗X, float_complex_indexed ∗Y)
- void **idxd_cmcmset** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_cisiset** (const int fold, const float_indexed ∗X, float_complex_indexed ∗Y)
- void **idxd_cmsmset** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_disetzero** (const int fold, double_indexed ∗X)
- void **idxd_dmsetzero** (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)
- void **idxd_zisetzero** (const int fold, double_complex_indexed ∗X)
- void **idxd_zmsetzero** (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)
- void **idxd_sisetzero** (const int fold, float_indexed ∗X)
- void **idxd_smsetzero** (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)
- void **idxd_cisetzero** (const int fold, float_complex_indexed ∗X)
- void **idxd_cmsetzero** (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)
- void **idxd_didiadd** (const int fold, const double_indexed ∗X, double_indexed ∗Y)
- void **idxd_dmdmadd** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_ziziadd** (const int fold, const double_complex_indexed ∗X, double_complex_indexed ∗Y)
- void **idxd_zmzmadd** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_sisiadd** (const int fold, const float_indexed ∗X, float_indexed ∗Y)
- void **idxd_smsmadd** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_ciciadd** (const int fold, const float_complex_indexed ∗X, float_complex_indexed ∗Y)
- void **idxd_cmcmadd** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_didiaddv** (const int fold, const int N, const double_indexed ∗X, const int incX, double_indexed ∗Y, const int incY)
- void **idxd_ziziaddv** (const int fold, const int N, const double_complex_indexed ∗X, const int incX, double_↩ complex_indexed ∗Y, const int incY)
- void **idxd_sisiaddv** (const int fold, const int N, const float_indexed ∗X, const int incX, float_indexed ∗Y, const int incY)
- void **idxd_ciciaddv** (const int fold, const int N, const float_complex_indexed ∗X, const int incX, float_↩ complex_indexed ∗Y, const int incY)
- void **idxd_didadd** (const int fold, const double X, double_indexed ∗Y)
- void **idxd_dmdadd** (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_zizadd** (const int fold, const void ∗X, double_complex_indexed ∗Y)
- void **idxd_zmzadd** (const int fold, const void ∗X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_sisadd** (const int fold, const float X, float_indexed ∗Y)
- void **idxd_smsadd** (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_cicadd** (const int fold, const void ∗X, float_complex_indexed ∗Y)
- void **idxd_cmcadd** (const int fold, const void ∗X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_didupdate** (const int fold, const double X, double_indexed ∗Y)
- void **idxd_dmdupdate** (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_zizupdate** (const int fold, const void ∗X, double_complex_indexed ∗Y)
- void **idxd_zmzupdate** (const int fold, const void ∗X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_zidupdate** (const int fold, const double X, double_complex_indexed ∗Y)

- void **idxd_zmdupdate** (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_sisupdate** (const int fold, const float X, float_indexed ∗Y)
- void **idxd_smsupdate** (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_cicupdate** (const int fold, const void ∗X, float_complex_indexed ∗Y)
- void **idxd_cmcupdate** (const int fold, const void ∗X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_cisupdate** (const int fold, const float X, float_complex_indexed ∗Y)
- void **idxd_cmsupdate** (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_diddeposit** (const int fold, const double X, double_indexed ∗Y)
- void **idxd_dmddeposit** (const int fold, const double X, double ∗priY, const int incpriY)
- void **idxd_zizdeposit** (const int fold, const void ∗X, double_complex_indexed ∗Y)
- void **idxd_zmzdeposit** (const int fold, const void ∗X, double ∗priY, const int incpriY)
- void **idxd_sisdeposit** (const int fold, const float X, float_indexed ∗Y)
- void **idxd_smsdeposit** (const int fold, const float X, float ∗priY, const int incpriY)
- void **idxd_cicdeposit** (const int fold, const void ∗X, float_complex_indexed ∗Y)
- void **idxd_cmcdeposit** (const int fold, const void ∗X, float ∗priY, const int incpriY)
- void **idxd_direnorm** (const int fold, double_indexed ∗X)
- void **idxd_dmrenorm** (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)
- void **idxd_zirenorm** (const int fold, double_complex_indexed ∗X)
- void **idxd_zmrenorm** (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)
- void **idxd_sirenorm** (const int fold, float_indexed ∗X)
- void **idxd_smrenorm** (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)
- void **idxd_cirenorm** (const int fold, float_complex_indexed ∗X)
- void **idxd_cmrenorm** (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)
- void **idxd_didconv** (const int fold, const double X, double_indexed ∗Y)
- void **idxd_dmdconv** (const int fold, const double X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_zizconv** (const int fold, const void ∗X, double_complex_indexed ∗Y)
- void **idxd_zmzconv** (const int fold, const void ∗X, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_sisconv** (const int fold, const float X, float_indexed ∗Y)
- void **idxd_smsconv** (const int fold, const float X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_cicconv** (const int fold, const void ∗X, float_complex_indexed ∗Y)
- void **idxd_cmcconv** (const int fold, const void ∗X, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- double **idxd_ddiconv** (const int fold, const double_indexed ∗X)
- double **idxd_ddmconv** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX)
- void **idxd_zziconv_sub** (const int fold, const double_complex_indexed ∗X, void ∗conv)
- void **idxd_zzmconv_sub** (const int fold, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, void ∗conv)
- float **idxd_ssiconv** (const int fold, const float_indexed ∗X)
- float **idxd_ssmconv** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX)
- void **idxd_cciconv_sub** (const int fold, const float_complex_indexed ∗X, void ∗conv)
- void **idxd_ccmconv_sub** (const int fold, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, void ∗conv)
- void **idxd_dinegate** (const int fold, double_indexed ∗X)
- void **idxd_dmnegate** (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)
- void **idxd_zinegate** (const int fold, double_complex_indexed ∗X)
- void **idxd_zmnegate** (const int fold, double ∗priX, const int incpriX, double ∗carX, const int inccarX)
- void **idxd_sinegate** (const int fold, float_indexed ∗X)
- void **idxd_smnegate** (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)
- void **idxd_cinegate** (const int fold, float_complex_indexed ∗X)
- void **idxd_cmnegate** (const int fold, float ∗priX, const int incpriX, float ∗carX, const int inccarX)
- double **idxd_dscale** (const double X)
- float **idxd_sscale** (const float X)

- void **idxd_dmdrescale** (const int fold, const double X, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_zmdrescale** (const int fold, const double X, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxd_smsrescale** (const int fold, const float X, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxd_cmsrescale** (const int fold, const float X, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- double **idxd_dmdmaddsq** (const int fold, const double scaleX, const double ∗priX, const int incpriX, const double ∗carX, const int inccarX, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- double **idxd_didiaddsq** (const int fold, const double scaleX, const double_indexed ∗X, const double scaleY, double_indexed ∗Y)
- float **idxd_smsmaddsq** (const int fold, const float scaleX, const float ∗priX, const int incpriX, const float ∗carX, const int inccarX, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- float **idxd_sisiaddsq** (const int fold, const float scaleX, const float_indexed ∗X, const float scaleY, float_↩ indexed ∗Y)
- double **idxd_ufp** (const double X)
- float **idxd_ufpf** (const float X)

2.1.1 Detailed Description

idxd.h defines the indexed types and the lower level functions associated with their use.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (∗`void`)
- s - float (`float`)
- c - complex float (∗`void`)
- di - indexed double (double_indexed)
- zi - indexed complex double (double_complex_indexed)
- si - indexed float (float_indexed)
- ci - indexed complex float (float_complex_indexed)
- dm - manually specified indexed double (`double`, `double`)
- zm - manually specified indexed complex double (`double`, `double`)
- sm - manually specified indexed float (`float`, `float`)
- cm - manually specified indexed complex float (`float`, `float`)

Throughout the library, complex types are specified via ∗`void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using indexed types is to obtain either more accurate or reproducible summation of floating point numbers. Indexed types are composed of several adjacent bins...

The parameter `fold` describes how many bins are used in the indexed types supplied to a subroutine. The maximum value for this parameter can be set in config.h. If you are unsure of what value to use for , we recommend 3. Note that the `fold` of indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all indexed functions that you use.Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

2.1.2 Macro Definition Documentation

2.1.2.1 #define DIWIDTH 40

Indexed double precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.2 #define idxd_DICAPACITY (idxd_DIENDURANCE*(1.0/DBL_EPSILON - 1.0))

Indexed double precision capacity.

The maximum number of double precision numbers that can be summed using indexed double precision. Applies also to indexed complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.3 #define idxd_DIENDURANCE (1 << (DBL_MANT_DIG - DIWIDTH - 2))

Indexed double precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to indexed complex double precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.4 #define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)

The maximum double precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.5 #define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)

Indexed double precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.6 #define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

Indexed double precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.7 #define idxd_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DIWIDTH + 1)))

Indexed double precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.8 #define idxd_SICAPACITY (idxd_SIENDURANCE*(1.0/FLT_EPSILON - 1.0))

Indexed single precision capacity.

The maximum number of single precision numbers that can be summed using indexed single precision. Applies also to indexed complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.9 #define idxd_SIENDURANCE (1 << (FLT_MANT_DIG - SIWIDTH - 2))

Indexed single precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to indexed complex single precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.10 #define idxd_SIMAXFOLD (idxd_SIMAXINDEX + 1)

The maximum single precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.11 #define idxd_SIMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SIWIDTH) - 1)

Indexed single precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.12 #define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

Indexed single precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.13 #define idxd_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SIWIDTH + 1)))

Indexed single precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.14 #define SIWIDTH 13

Indexed single precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.3 Typedef Documentation

2.1.3.1 typedef double double_complex_indexed

The indexed complex double datatype.

To allocate a double_complex_indexed, call idxd_zialloc()

Warning

A double_complex_indexed is, under the hood, an array of `double`. Therefore, if you have defined an array of double_complex_indexed, you must index it by multiplying the index into the array by the number of underlying `double` that make up the double_complex_indexed. This number can be obtained by a call to idxd_zinum()

2.1.3.2 typedef double double_indexed

The indexed double datatype.

To allocate a double_indexed, call idxd_dialloc()

Warning

A double_indexed is, under the hood, an array of `double`. Therefore, if you have defined an array of double↵ _indexed, you must index it by multiplying the index into the array by the number of underlying `double` that make up the double_indexed. This number can be obtained by a call to idxd_dinum()

2.1.3.3 typedef float float_complex_indexed

The indexed complex float datatype.

To allocate a float_complex_indexed, call idxd_cialloc()

Warning

A float_complex_indexed is, under the hood, an array of `float`. Therefore, if you have defined an array of float_complex_indexed, you must index it by multiplying the index into the array by the number of underlying `float` that make up the float_complex_indexed. This number can be obtained by a call to idxd_cinum()

2.1.3.4 typedef float float_indexed

The indexed float datatype.

To allocate a float_indexed, call idxd_sialloc()

Warning

A float_indexed is, under the hood, an array of `float`. Therefore, if you have defined an array of float_↵ indexed, you must index it by multiplying the index into the array by the number of underlying `float` that make up the float_indexed. This number can be obtained by a call to idxd_sinum()

2.2 include/idxdBLAS.h File Reference

idxdBLAS.h defines BLAS Methods that operate on indexed types.

```
#include "idxd.h"
#include "reproBLAS.h"
#include <complex.h>
```

Functions

- float **idxdBLAS_samax** (const int N, const float ∗X, const int incX)
- double **idxdBLAS_damax** (const int N, const double ∗X, const int incX)
- void **idxdBLAS_camax_sub** (const int N, const void ∗X, const int incX, void ∗amax)
- void **idxdBLAS_zamax_sub** (const int N, const void ∗X, const int incX, void ∗amax)
- float **idxdBLAS_samaxm** (const int N, const float ∗X, const int incX, const float ∗Y, const int incY)
- double **idxdBLAS_damaxm** (const int N, const double ∗X, const int incX, const double ∗Y, const int incY)
- void **idxdBLAS_camaxm_sub** (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗amaxm)

- void **idxdBLAS_zamaxm_sub** (const int N, const void ∗X, const int incX, const void ∗Y, const int incY, void ∗amaxm)
- void **idxdBLAS_didsum** (const int fold, const int N, const double ∗X, const int incX, double_indexed ∗Y)
- void **idxdBLAS_dmdsum** (const int fold, const int N, const double ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxdBLAS_didasum** (const int fold, const int N, const double ∗X, const int incX, double_indexed ∗Y)
- void **idxdBLAS_dmdasum** (const int fold, const int N, const double ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- double **idxdBLAS_didssq** (const int fold, const int N, const double ∗X, const int incX, const double scaleY, double_indexed ∗Y)
- double **idxdBLAS_dmdssq** (const int fold, const int N, const double ∗X, const int incX, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxdBLAS_diddot** (const int fold, const int N, const double ∗X, const int incX, const double ∗Y, const int incY, double_indexed ∗Z)
- void **idxdBLAS_dmddot** (const int fold, const int N, const double ∗X, const int incX, const double ∗Y, const int incY, double ∗manZ, const int incmanZ, double ∗carZ, const int inccarZ)
- void **idxdBLAS_dizsum** (const int fold, const int N, const void ∗X, const int incX, double_indexed ∗Y)
- void **idxdBLAS_dmzsum** (const int fold, const int N, const void ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxdBLAS_dizasum** (const int fold, const int N, const void ∗X, const int incX, double_indexed ∗Y)
- void **idxdBLAS_dmzasum** (const int fold, const int N, const void ∗X, const int incX, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- double **idxdBLAS_dizssq** (const int fold, const int N, const void ∗X, const int incX, const double scaleY, double_indexed ∗Y)
- double **idxdBLAS_dmzssq** (const int fold, const int N, const void ∗X, const int incX, const double scaleY, double ∗priY, const int incpriY, double ∗carY, const int inccarY)
- void **idxdBLAS_zizdotu** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double_indexed ∗Z)
- void **idxdBLAS_zmzdotu** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double ∗manZ, const int incmanZ, double ∗carZ, const int inccarZ)
- void **idxdBLAS_zizdotc** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double_indexed ∗Z)
- void **idxdBLAS_zmzdotc** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, double ∗manZ, const int incmanZ, double ∗carZ, const int inccarZ)
- void **idxdBLAS_sissum** (const int fold, const int N, const float ∗X, const int incX, float_indexed ∗Y)
- void **idxdBLAS_smssum** (const int fold, const int N, const float ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxdBLAS_sisasum** (const int fold, const int N, const float ∗X, const int incX, float_indexed ∗Y)
- void **idxdBLAS_smsasum** (const int fold, const int N, const float ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- float **idxdBLAS_sisssq** (const int fold, const int N, const float ∗X, const int incX, const float scaleY, float_↵ indexed ∗Y)
- float **idxdBLAS_smsssq** (const int fold, const int N, const float ∗X, const int incX, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxdBLAS_sisdot** (const int fold, const int N, const float ∗X, const int incX, const float ∗Y, const int incY, float_indexed ∗Z)
- void **idxdBLAS_smsdot** (const int fold, const int N, const float ∗X, const int incX, const float ∗Y, const int incY, float ∗manZ, const int incmanZ, float ∗carZ, const int inccarZ)
- void **idxdBLAS_cicsum** (const int fold, const int N, const void ∗X, const int incX, float_indexed ∗Y)
- void **idxdBLAS_cmcsum** (const int fold, const int N, const void ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxdBLAS_sicasum** (const int fold, const int N, const void ∗X, const int incX, float_indexed ∗Y)
- void **idxdBLAS_smcasum** (const int fold, const int N, const void ∗X, const int incX, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- float **idxdBLAS_sicssq** (const int fold, const int N, const void ∗X, const int incX, const float scaleY, float_↵ indexed ∗Y)

- float **idxdBLAS_smcssq** (const int fold, const int N, const void ∗X, const int incX, const float scaleY, float ∗priY, const int incpriY, float ∗carY, const int inccarY)
- void **idxdBLAS_cicdotu** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float_indexed ∗Z)
- void **idxdBLAS_cmcdotu** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float ∗manZ, const int incmanZ, float ∗carZ, const int inccarZ)
- void **idxdBLAS_cicdotc** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float_indexed ∗Z)
- void **idxdBLAS_cmcdotc** (const int fold, const int N, const void ∗X, const int incX, const void ∗Y, const int incY, float ∗manZ, const int incmanZ, float ∗carZ, const int inccarZ)
- void **idxdBLAS_didgemv** (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, double_indexed ∗Y, const int incY)
- void **idxdBLAS_didgemm** (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const int lda, const double ∗B, const int ldb, double_indexed ∗C, const int ldc)
- void **idxdBLAS_sisgemv** (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float ∗A, const int lda, const float ∗X, const int incX, float_indexed ∗Y, const int incY)
- void **idxdBLAS_sisgemm** (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float ∗A, const int lda, const float ∗B, const int ldb, float_indexed ∗C, const int ldc)
- void **idxdBLAS_zizgemv** (const int fold, const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, double_complex_indexed ∗Y, const int incY)
- void **idxdBLAS_zizgemm** (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, double_complex_indexed ∗C, const int ldc)
- void **idxdBLAS_cicgemv** (const int fold, const char Order, const char TransA, const int M, const int N, const void ∗alpha, const void ∗A, const int lda, const void ∗X, const int incX, float_complex_indexed ∗Y, const int incY)
- void **idxdBLAS_cicgemm** (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void ∗alpha, const void ∗A, const int lda, const void ∗B, const int ldb, float_complex_indexed ∗C, const int ldc)

2.2.1 Detailed Description

idxdBLAS.h defines BLAS Methods that operate on indexed types.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (∗`void`)
- s - float (`float`)
- c - complex float (∗`void`)
- di - indexed double (double_indexed)
- zi - indexed complex double (double_complex_indexed)
- si - indexed float (float_indexed)

- ci - indexed complex float (float_complex_indexed)
- dm - manually specified indexed double (`double`, `double`)
- zm - manually specified indexed complex double (`double`, `double`)
- sm - manually specified indexed float (`float`, `float`)
- cm - manually specified indexed complex float (`float`, `float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using indexed types is to obtain either more accurate or reproducible summation of floating point numbers. Indexed types are composed of several adjacent bins...

The parameter `fold` describes how many bins are used in the indexed types supplied to a subroutine. The maximum value for this parameter can be set in config.h. If you are unsure of what value to use for , we recommend 3. Note that the `fold` of indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all indexed functions that you use.Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

Index