

ReproBLAS

Generated by Doxygen 1.8.9.1

Tue Apr 28 2015 17:56:40



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	dlComplex_ Struct Reference . . . . .	5
3.2	ldouble_ Struct Reference . . . . .	5
3.3	lfloat_ Struct Reference . . . . .	5
3.4	slComplex_ Struct Reference . . . . .	6
<b>4</b>	<b>File Documentation</b>	<b>7</b>
4.1	include/indexed.h File Reference . . . . .	7
4.1.1	Detailed Description . . . . .	12
4.1.2	Typedef Documentation . . . . .	12
4.1.2.1	double_complex_indexed . . . . .	12
4.1.2.2	double_indexed . . . . .	13
4.1.2.3	float_complex_indexed . . . . .	13
4.1.2.4	float_indexed . . . . .	13
4.1.3	Function Documentation . . . . .	13
4.1.3.1	cciconv_sub . . . . .	13
4.1.3.2	cialloc . . . . .	14
4.1.3.3	cicadd . . . . .	15
4.1.3.4	cicconv . . . . .	15
4.1.3.5	cicdeposit . . . . .	16
4.1.3.6	ciciadd . . . . .	16
4.1.3.7	ciciset . . . . .	16
4.1.3.8	cicupdate . . . . .	17
4.1.3.9	cinegate . . . . .	17
4.1.3.10	cinum . . . . .	17
4.1.3.11	ciprint . . . . .	18

4.1.3.12	cirenorm	18
4.1.3.13	cisetzero	18
4.1.3.14	cisiset	19
4.1.3.15	cisize	19
4.1.3.16	cisupdate	19
4.1.3.17	dbound	20
4.1.3.18	ddiconv	20
4.1.3.19	dialloc	20
4.1.3.20	dicapacity	21
4.1.3.21	didadd	21
4.1.3.22	didconv	21
4.1.3.23	diddeposit	22
4.1.3.24	didiadd	22
4.1.3.25	didiset	23
4.1.3.26	didupdate	23
4.1.3.27	dindex	23
4.1.3.28	dinegate	24
4.1.3.29	dinum	24
4.1.3.30	diprint	24
4.1.3.31	direnorm	25
4.1.3.32	disetzero	25
4.1.3.33	disize	25
4.1.3.34	diwidth	26
4.1.3.35	sbound	26
4.1.3.36	sialloc	26
4.1.3.37	sicapacity	27
4.1.3.38	sindex	27
4.1.3.39	sinegate	28
4.1.3.40	sinum	28
4.1.3.41	siprint	28
4.1.3.42	sirenorm	29
4.1.3.43	sisadd	30
4.1.3.44	sisconv	30
4.1.3.45	sisdeposit	31
4.1.3.46	sisetzero	31
4.1.3.47	sisiadd	31
4.1.3.48	sisiset	32
4.1.3.49	sisize	32
4.1.3.50	sisupdate	33
4.1.3.51	siwidth	34

4.1.3.52	ssiconv	34
4.1.3.53	zialloc	34
4.1.3.54	zidiset	35
4.1.3.55	zidupdate	35
4.1.3.56	zinegate	36
4.1.3.57	zinum	37
4.1.3.58	ziprint	37
4.1.3.59	zirenorm	37
4.1.3.60	zisetzero	38
4.1.3.61	zisize	38
4.1.3.62	zizadd	38
4.1.3.63	zizconv	39
4.1.3.64	zizdeposit	39
4.1.3.65	ziziadd	40
4.1.3.66	ziziset	40
4.1.3.67	zizupdate	40
4.1.3.68	zziconv_sub	41
<b>Index</b>		<b>43</b>



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">dlComplex_</a> . . . . .	5
<a href="#">ldouble_</a> . . . . .	5
<a href="#">lfloat_</a> . . . . .	5
<a href="#">slComplex_</a> . . . . .	6





## Chapter 2

# File Index

### 2.1 File List

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

include/[indexed.h](#)

[Indexed.h](#) defines the indexed types and the lower level functions associated with their use . . . [7](#)



## Chapter 3

# Data Structure Documentation

### 3.1 dlComplex\_ Struct Reference

#### Data Fields

- double **m** [2 \*DEFAULT\_FOLD]
- double **c** [2 \*DEFAULT\_FOLD]

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

- include/[indexed.h](#)

### 3.2 Idouble\_ Struct Reference

#### Data Fields

- double **m** [DEFAULT\_FOLD]
- double **c** [DEFAULT\_FOLD]

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

- include/[indexed.h](#)

### 3.3 Ifloat\_ Struct Reference

#### Data Fields

- float **m** [DEFAULT\_FOLD]
- float **c** [DEFAULT\_FOLD]

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

- include/[indexed.h](#)

### 3.4 slComplex\_ Struct Reference

#### Data Fields

- float **m** [2 \*DEFAULT\_FOLD]
- float **c** [2 \*DEFAULT\_FOLD]

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

- include/[indexed.h](#)

## Chapter 4

# File Documentation

### 4.1 include/indexed.h File Reference

[indexed.h](#) defines the indexed types and the lower level functions associated with their use.

```
#include <complex.h>
#include <math.h>
#include <stddef.h>
```

#### Data Structures

- struct [ldouble\\_](#)
- struct [dlComplex\\_](#)
- struct [lfloat\\_](#)
- struct [slComplex\\_](#)

#### Macros

- #define **DEFAULT\_FOLD** 3
- #define **ldouble** [l\\_double](#)
- #define **lfloat** [l\\_float](#)
- #define **dlcomplex** [l\\_double\\_Complex](#)
- #define **slcomplex** [l\\_float\\_Complex](#)

#### 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.*
- typedef struct [ldouble\\_ l\\_double](#)
- typedef struct [dlComplex\\_ l\\_double\\_Complex](#)
- typedef struct [lfloat\\_ l\\_float](#)
- typedef struct [slComplex\\_ l\\_float\\_Complex](#)

## Functions

- `size_t disize` (const int fold)  
*indexed double precision size*
- `size_t zisize` (const int fold)  
*indexed complex double precision size*
- `size_t sisize` (const int fold)  
*indexed single precision size*
- `size_t cisize` (const int fold)  
*indexed complex single precision size*
- `double_indexed * dialloc` (const int fold)  
*indexed double precision allocation*
- `double_complex_indexed * zialloc` (const int fold)  
*indexed complex double precision allocation*
- `float_indexed * sialloc` (const int fold)  
*indexed single precision allocation*
- `float_complex_indexed * cialloc` (const int fold)  
*indexed complex single precision allocation*
- `int dinum` (const int fold)  
*indexed double precision size*
- `int zinum` (const int fold)  
*indexed complex double precision size*
- `int sinum` (const int fold)  
*indexed single precision size*
- `int cinum` (const int fold)  
*indexed complex single precision size*
- `int diwidth` ()  
*Get indexed double precision bin width.*
- `int siwidth` ()  
*Get indexed single precision bin width.*
- `int dicapacity` ()  
*Get indexed double precision deposit capacity.*
- `int sicapacity` ()  
*Get indexed single precision deposit capacity.*
- `double dbound` (const int index)  
*Get double precision bound corresponding to index.*
- `void dmbound` (const int fold, const int index, double \*repX, const int increpX, double \*carX, const int inccarX)
- `int dindex` (const double X)  
*Get index of double precision.*
- `int dmindex` (const double \*repX)
- `float sbound` (const int index)  
*Get single precision bound corresponding to index.*
- `void smbnd` (const int fold, const int index, float \*repX, const int increpX, float \*carX, const int inccarX)
- `int sindex` (const float X)  
*Get index of single precision.*
- `int smindex` (const float \*repX)
- `void ciprint` (const int fold, const `float_complex_indexed` \*X)  
*Print indexed complex single precision.*
- `void cmprint` (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX)
- `void diprint` (const int fold, const `double_indexed` \*X)  
*Print indexed double precision.*

- void **dmprint** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX)
- void **siprint** (const int fold, const [float\\_indexed](#) \*X)  
*Print indexed single precision.*
- void **smprint** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX)
- void **ziprint** (const int fold, const [double\\_complex\\_indexed](#) \*X)  
*Print indexed complex double precision.*
- void **zmpprint** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX)
- void **didiset** (const int fold, const [double\\_indexed](#) \*X, [double\\_indexed](#) \*Y)  
*Set indexed double precision ( $Y = X$ )*
- void **dmdmset** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **ziziset** (const int fold, const [double\\_complex\\_indexed](#) \*X, [double\\_complex\\_indexed](#) \*Y)  
*Set indexed complex double precision ( $Y = X$ )*
- void **zmzmset** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **zidiset** (const int fold, const [double\\_indexed](#) \*X, [double\\_complex\\_indexed](#) \*Y)  
*Set indexed complex double precision to indexed double precision ( $Y = X$ )*
- void **zmdmset** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **sisiset** (const int fold, const [float\\_indexed](#) \*X, [float\\_indexed](#) \*Y)  
*Set indexed single precision ( $Y = X$ )*
- void **smsmset** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **ciciset** (const int fold, const [float\\_complex\\_indexed](#) \*X, [float\\_complex\\_indexed](#) \*Y)  
*Set indexed complex single precision ( $Y = X$ )*
- void **cmcmset** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **cisiset** (const int fold, const [float\\_indexed](#) \*X, [float\\_complex\\_indexed](#) \*Y)  
*Set indexed complex single precision to indexed single precision ( $Y = X$ )*
- void **cmsmset** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **disetzero** (const int fold, [double\\_indexed](#) \*X)  
*Set indexed double precision to 0 ( $X = 0$ )*
- void **dmsetzero** (const int fold, double \*repX, const int increpX, double \*carX, const int inccarX)
- void **zisetzero** (const int fold, [double\\_complex\\_indexed](#) \*X)  
*Set indexed double precision to 0 ( $X = 0$ )*
- void **zmsetzero** (const int fold, double \*repX, const int increpX, double \*carX, const int inccarX)
- void **sisetzero** (const int fold, [float\\_indexed](#) \*X)  
*Set indexed single precision to 0 ( $X = 0$ )*
- void **smsetzero** (const int fold, float \*repX, const int increpX, float \*carX, const int inccarX)
- void **cisetzero** (const int fold, [float\\_complex\\_indexed](#) \*X)  
*Set indexed single precision to 0 ( $X = 0$ )*
- void **cmsetzero** (const int fold, float \*repX, const int increpX, float \*carX, const int inccarX)
- void **didiadd** (const int fold, const [double\\_indexed](#) \*X, [double\\_indexed](#) \*Y)  
*Add indexed double precision ( $Y += X$ )*
- void **dmdmadd** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **ziziadd** (const int fold, const [double\\_complex\\_indexed](#) \*X, [double\\_complex\\_indexed](#) \*Y)  
*Add indexed complex double precision ( $Y += X$ )*
- void **zmzmadd** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **sisiadd** (const int fold, const [float\\_indexed](#) \*X, [float\\_indexed](#) \*Y)  
*Add indexed single precision ( $Y += X$ )*

- void **smsmadd** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **ciciadd** (const int fold, const [float\\_complex\\_indexed](#) \*X, [float\\_complex\\_indexed](#) \*Y)
  - Add indexed complex single precision (Y += X)*
- void **cmcmadd** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **didadd** (const int fold, const double X, [double\\_indexed](#) \*Y)
  - Add double precision to indexed double precision (Y += X)*
- void **dmdadd** (const int fold, const double X, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **zizadd** (const int fold, const void \*X, [double\\_complex\\_indexed](#) \*Y)
  - Add complex double precision to indexed complex double precision (Y += X)*
- void **zmzadd** (const int fold, const void \*X, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **sisadd** (const int fold, const float X, [float\\_indexed](#) \*Y)
  - Add single precision to indexed single precision (Y += X)*
- void **smsadd** (const int fold, const float X, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **cicadd** (const int fold, const void \*X, [float\\_complex\\_indexed](#) \*Y)
  - Add complex single precision to indexed complex single precision (Y += X)*
- void **cmcadd** (const int fold, const void \*X, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **didupdate** (const int fold, const double X, [double\\_indexed](#) \*Y)
  - Update indexed double precision with double precision (X -> Y)*
- void **dmdupdate** (const int fold, const double X, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **zizupdate** (const int fold, const void \*X, [double\\_complex\\_indexed](#) \*Y)
  - Update indexed complex double precision with complex double precision (X -> Y)*
- void **zmzupdate** (const int fold, const void \*X, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **zidupdate** (const int fold, const double X, [double\\_complex\\_indexed](#) \*Y)
  - Update indexed complex double precision with double precision (X -> Y)*
- void **zmdupdate** (const int fold, const double X, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **sisupdate** (const int fold, const float X, [float\\_indexed](#) \*Y)
  - Update indexed single precision with single precision (X -> Y)*
- void **smsupdate** (const int fold, const float X, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **cicupdate** (const int fold, const void \*X, [float\\_complex\\_indexed](#) \*Y)
  - Update indexed complex single precision with complex single precision (X -> Y)*
- void **cmcupdate** (const int fold, const void \*X, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **cisupdate** (const int fold, const float X, [float\\_complex\\_indexed](#) \*Y)
  - Update indexed complex single precision with single precision (X -> Y)*
- void **cmsupdate** (const int fold, const float X, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **diddeposit** (const int fold, const double X, [double\\_indexed](#) \*Y)
  - Add double precision to suitably indexed indexed double precision (Y += X)*
- void **dmddeposit** (const int fold, const double X, double \*repY, const int increpY)
- void **zizdeposit** (const int fold, const void \*X, [double\\_complex\\_indexed](#) \*Y)
  - Add complex double precision to suitably indexed manually specified indexed complex double precision (Y += X)*
- void **zmzdeposit** (const int fold, const void \*X, double \*repY, const int increpY)
- void **sisdeposit** (const int fold, const float X, [float\\_indexed](#) \*Y)
  - Add single precision to suitably indexed indexed single precision (Y += X)*
- void **smsdeposit** (const int fold, const float X, float \*repY, const int increpY)
- void **cicdeposit** (const int fold, const void \*X, [float\\_complex\\_indexed](#) \*Y)
  - Add complex single precision to suitably indexed manually specified indexed complex single precision (Y += X)*
- void **cmcdeposit** (const int fold, const void \*X, float \*repY, const int increpY)
- void **direnorm** (const int fold, [double\\_indexed](#) \*X)
  - Renormalize indexed double precision.*



- void **dmrenorm** (const int fold, double \*repX, const int increpX, double \*carX, const int inccarX)
- void **zirenorm** (const int fold, [double\\_complex\\_indexed](#) \*X)  
*Renormalize indexed complex double precision.*
- void **zmrenorm** (const int fold, double \*repX, const int increpX, double \*carX, const int inccarX)
- void **sirenorm** (const int fold, [float\\_indexed](#) \*X)  
*Renormalize indexed single precision.*
- void **smrenorm** (const int fold, float \*repX, const int increpX, float \*carX, const int inccarX)
- void **cirenorm** (const int fold, [float\\_complex\\_indexed](#) \*X)  
*Renormalize indexed complex single precision.*
- void **cmrenorm** (const int fold, float \*repX, const int increpX, float \*carX, const int inccarX)
- void **didconv** (const int fold, const double X, [double\\_indexed](#) \*Y)  
*Convert double precision to indexed double precision (X -> Y)*
- void **dmdconv** (const int fold, const double X, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **zizconv** (const int fold, const void \*X, [double\\_complex\\_indexed](#) \*Y)  
*Convert complex double precision to indexed complex double precision (X -> Y)*
- void **zmzconv** (const int fold, const void \*X, double \*repY, const int increpY, double \*carY, const int inccarY)
- void **sisconv** (const int fold, const float X, [float\\_indexed](#) \*Y)  
*Convert single precision to indexed single precision (X -> Y)*
- void **smsconv** (const int fold, const float X, float \*repY, const int increpY, float \*carY, const int inccarY)
- void **cicconv** (const int fold, const void \*X, [float\\_complex\\_indexed](#) \*Y)  
*Convert complex single precision to indexed complex single precision (X -> Y)*
- void **cmcconv** (const int fold, const void \*X, float \*repY, const int increpY, float \*carY, const int inccarY)
- double **ddiconv** (const int fold, const [double\\_indexed](#) \*X)  
*Convert indexed double precision to double precision (X -> Y)*
- double **ddmconv** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX)
- void **zziconv\_sub** (const int fold, const [double\\_complex\\_indexed](#) \*X, void \*conv)  
*Convert indexed complex double precision to complex double precision (X -> Y)*
- void **zzmconv\_sub** (const int fold, const double \*repX, const int increpX, const double \*carX, const int inccarX, void \*conv)
- float **ssiconv** (const int fold, const [float\\_indexed](#) \*X)  
*Convert indexed single precision to single precision (X -> Y)*
- float **ssmconv** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX)
- void **cciconv\_sub** (const int fold, const [float\\_complex\\_indexed](#) \*X, void \*conv)  
*Convert indexed complex single precision to complex single precision (X -> Y)*
- void **ccmconv\_sub** (const int fold, const float \*repX, const int increpX, const float \*carX, const int inccarX, void \*conv)
- void **dinegate** (const int fold, [double\\_indexed](#) \*X)  
*Negate indexed double precision (X = -X)*
- void **dmnegate** (const int fold, double \*repX, const int increpX, double \*carX, const int inccarX)
- void **zinegate** (const int fold, [double\\_complex\\_indexed](#) \*X)  
*Negate indexed complex double precision (X = -X)*
- void **zmnegate** (const int fold, double \*repX, const int increpX, double \*carX, const int inccarX)
- void **sinegate** (const int fold, [float\\_indexed](#) \*X)  
*Negate indexed single precision (X = -X)*
- void **smnegate** (const int fold, float \*repX, const int increpX, float \*carX, const int inccarX)
- void **cinegate** (const int fold, [float\\_complex\\_indexed](#) \*X)  
*Negate indexed complex single precision (X = -X)*
- void **cmnegate** (const int fold, float \*repX, const int increpX, float \*carX, const int inccarX)
- double **ufp** (const double X)
- float **ufpf** (const float X)

### 4.1.1 Detailed Description

[indexed.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.

### 4.1.2 Typedef Documentation

#### 4.1.2.1 typedef `double double_complex_indexed`

The indexed complex double datatype.

To allocate a [double\\_complex\\_indexed](#), call [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 [zinum\(\)](#)

## 4.1.2.2 typedef double double\_indexed

The indexed double datatype.

To allocate a `double_indexed`, call `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 `dinum()`

## 4.1.2.3 typedef float float\_complex\_indexed

The indexed complex float datatype.

To allocate a `float_complex_indexed`, call `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 `cinum()`

## 4.1.2.4 typedef float float\_indexed

The indexed float datatype.

To allocate a `float_indexed`, call `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 `sinum()`

## 4.1.3 Function Documentation

## 4.1.3.1 void cciconv\_sub ( const int fold, const float\_complex\_indexed \* X, void \* conv )

Convert indexed complex single precision to complex single precision (X -> Y)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>conv</i>	scalar return

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

#### 4.1.3.2 `float_complex_indexed*` `calloc` ( `const int fold` )

indexed complex single precision allocation

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

a freshly allocated indexed type. (free with `free()`)

## Author

Peter Ahrens

## Date

27 Apr 2015

**4.1.3.3 void cicadd ( const int *fold*, const void \* *X*, float\_complex\_indexed \* *Y* )**

Add complex single precision to indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type *Y*

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.4 void cicconv ( const int *fold*, const void \* *X*, float\_complex\_indexed \* *Y* )**

Convert complex single precision to indexed complex single precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

#### 4.1.3.5 void cicdeposit ( const int *fold*, const void \* *X*, float\_complex\_indexed \* *Y* )

Add complex single precision to suitably indexed manually specified indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$  where the index of  $Y$  is larger than the index of  $X$

##### Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [cicupdate\(\)](#) on  $Y$  with the maximum absolute value of any elements you wish to deposit in  $Y$ , you can call this method to deposit a maximum of [sicapacity\(\)](#) elements into  $Y$ . After calling [cicdeposit\(\)](#) on an indexed type, you must renormalize the indexed type with [cirenorm\(\)](#).

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>repY</i>	$Y$ 's rep vector
<i>increpY</i>	stride within $Y$ 's rep vector (use every $increpY$ th element)

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

#### 4.1.3.6 void ciciadd ( const int *fold*, const float\_complex\_indexed \* *X*, float\_complex\_indexed \* *Y* )

Add indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar $X$
<i>Y</i>	indexed scalar $Y$

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

#### 4.1.3.7 void ciciset ( const int *fold*, const float\_complex\_indexed \* *X*, float\_complex\_indexed \* *Y* )

Set indexed complex single precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.8 void cicupdate ( const int *fold*, const void \* *X*, float\_complex\_indexed \* *Y* )

Update indexed complex single precision with complex single precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of *X* respectively.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.9 void cinegate ( const int *fold*, float\_complex\_indexed \* *X* )

Negate indexed complex single precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.10 int cinum ( const int *fold* )

indexed complex single precision size

**Parameters**

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

**Returns**

the size (in `float`) of the indexed type

**Author**

Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.11 void ciprint ( const int *fold*, const float\_complex\_indexed \* *X* )**

Print indexed complex single precision.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.12 void cirenorm ( const int *fold*, float\_complex\_indexed \* *X* )**

Renormalize indexed complex single precision.

Renormalization keeps the rep vector within the necessary bounds by shifting over to the carry vector

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.13 void cisetzero ( const int *fold*, float\_complex\_indexed \* *X* )**

Set indexed single precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$



## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.14 void cisiset ( const int *fold*, const float\_indexed \* *X*, float\_complex\_indexed \* *Y* )**

Set indexed complex single precision to indexed single precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.15 size\_t cisize ( const int *fold* )**

indexed complex single precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in bytes) of the indexed type

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.16 void cisupdate ( const int *fold*, const float *X*, float\_complex\_indexed \* *Y* )**

Update indexed complex single precision with single precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value less than X

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.17 `double dbound ( const int index )`

Get double precision bound corresponding to index.

## Parameters

<i>index</i>	index
--------------	-------

## Returns

bound (bin)

## Author

Peter Ahrens

## Date

27 Apr 2015

4.1.3.18 `double ddiconv ( const int fold, const double_indexed * X )`

Convert indexed double precision to double precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Returns

scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.19 `double_indexed* dialloc ( const int fold )`

indexed double precision allocation

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

a freshly allocated indexed type. (free with `free()`)

## Author

Peter Ahrens

## Date

27 Apr 2015

4.1.3.20 `int dicapacity ( )`

Get indexed double precision deposit capacity.

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

## Returns

deposit capacity

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.21 `void didadd ( const int fold, const double X, double_indexed * Y )`

Add double precision to indexed double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>Y</i>	indexed scalar $Y$

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.22 `void didconv ( const int fold, const double X, double_indexed * Y )`

Convert double precision to indexed double precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.23 void diddeposit ( const int *fold*, const double *X*, double\_indexed \* *Y* )

Add double precision to suitably indexed indexed double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type *Y* where the index of *Y* is larger than the index of *X*

## Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [didupdate\(\)](#) on *Y* with the maximum absolute value of any elements you wish to deposit in *Y*, you can call this method to deposit a maximum of [dicapacity\(\)](#) elements into *Y*. After calling [diddeposit\(\)](#) on an indexed type, you must renormalize the indexed type with [direnorm\(\)](#).

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.24 void didiadd ( const int *fold*, const double\_indexed \* *X*, double\_indexed \* *Y* )

Add indexed double precision ( $Y += X$ )

Performs the operation  $Y += X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

Y	indexed scalar Y
---	------------------

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.25 void didiset ( const int *fold*, const double\_indexed \* X, double\_indexed \* Y )**

Set indexed double precision ( $Y = X$ )

Performs the operation  $Y = X$

**Parameters**

<i>fold</i>	the fold of the indexed types
X	indexed scalar X
Y	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.26 void didupdate ( const int *fold*, const double X, double\_indexed \* Y )**

Update indexed double precision with double precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value less than X

**Parameters**

<i>fold</i>	the fold of the indexed types
X	scalar X
Y	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.27 int dindex ( const double X )**

Get index of double precision.

The index of a non-indexed type is the smallest index an indexed type would need to have to sum it reproducibly. Higher indices correspond to smaller bins.

## Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

## Returns

*X*'s index

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.28 void dinegate ( const int *fold*, double\_indexed \* *X* )

Negate indexed double precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.29 int dinum ( const int *fold* )

indexed double precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in double) of the indexed type

## Author

Peter Ahrens

## Date

27 Apr 2015

4.1.3.30 void diprint ( const int *fold*, const double\_indexed \* *X* )

Print indexed double precision.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.31 void direnorm ( const int *fold*, double\_indexed \* *X* )**

Renormalize indexed double precision.

Renormalization keeps the rep vector within the necessary bounds by shifting over to the carry vector

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.32 void disetzero ( const int *fold*, double\_indexed \* *X* )**

Set indexed double precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.33 size\_t disize ( const int *fold* )**

indexed double precision size

**Parameters**

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

**Returns**

the size (in bytes) of the indexed type

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.34 int diwidth ( )**

Get indexed double precision bin width.

**Returns**

bin width (in bits)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.35 float sbound ( const int *index* )**

Get single precision bound corresponding to index.

**Parameters**

<i>index</i>	index
--------------	-------

**Returns**

bound (bin)

**Author**

Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.36 float\_indexed\* sialloc ( const int *fold* )**

indexed single precision allocation



## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

a freshly allocated indexed type. (free with `free()`)

## Author

Peter Ahrens

## Date

27 Apr 2015

4.1.3.37 `int sicapacity ( )`

Get indexed single precision deposit capacity.

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

## Returns

deposit capacity

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.38 `int sindex ( const float X )`

Get index of single precision.

The index of a non-indexed type is the smallest index an indexed type would need to have to sum it reproducibly. Higher indices correspond to smaller bins.

## Parameters

<i>X</i>	scalar X
----------	----------

## Returns

X's index

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

#### 4.1.3.39 void sinegate ( const int *fold*, float\_indexed \* *X* )

Negate indexed single precision ( $X = -X$ )

Performs the operation  $X = -X$

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

#### 4.1.3.40 int sinum ( const int *fold* )

indexed single precision size

##### Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

##### Returns

the size (in `float`) of the indexed type

##### Author

Peter Ahrens

##### Date

27 Apr 2015

#### 4.1.3.41 void siprint ( const int *fold*, const float\_indexed \* *X* )

Print indexed single precision.

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

4.1.3.42 void sirenorm ( const int *fold*, float\_indexed \* *X* )

Renormalize indexed single precision.

Renormalization keeps the rep vector within the necessary bounds by shifting over to the carry vector

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.43 void sisadd ( const int *fold*, const float *X*, float\_indexed \* *Y* )**

Add single precision to indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type *Y*

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.44 void sisconv ( const int *fold*, const float *X*, float\_indexed \* *Y* )**

Convert single precision to indexed single precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.45 void sisdeposit ( const int *fold*, const float *X*, float\_indexed \* *Y* )**

Add single precision to suitably indexed indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$  where the index of  $Y$  is larger than the index of  $X$

**Note**

This routine was provided as a means of allowing the you to optimize your code. After you have called [sisupdate\(\)](#) on  $Y$  with the maximum absolute value of any elements you wish to deposit in  $Y$ , you can call this method to deposit a maximum of [sicapacity\(\)](#) elements into  $Y$ . After calling [sisdeposit\(\)](#) on an indexed type, you must renormalize the indexed type with [sirenorm\(\)](#).

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>Y</i>	indexed scalar $Y$

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.46 void sisetzero ( const int *fold*, float\_indexed \* *X* )**

Set indexed single precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar $X$

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.47 void sisiadd ( const int *fold*, const float\_indexed \* *X*, float\_indexed \* *Y* )**

Add indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.48 void sisiset ( const int *fold*, const float\_indexed \* *X*, float\_indexed \* *Y* )**

Set indexed single precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.49 size\_t ssize ( const int *fold* )**

indexed single precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in bytes) of the indexed type

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.50 void sisupdate ( const int *fold*, const float *X*, float\_indexed \* *Y* )

Update indexed single precision with single precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.51 `int siwidth ( )`

Get indexed single precision bin width.

## Returns

bin width (in bits)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.52 `float ssiconv ( const int fold, const float_indexed * X )`

Convert indexed single precision to single precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Returns

scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.53 `double_complex_indexed* zialloc ( const int fold )`

indexed complex double precision allocation



## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

a freshly allocated indexed type. (free with `free()`)

## Author

Peter Ahrens

## Date

27 Apr 2015

#### 4.1.3.54 void `zidiset` ( const int *fold*, const `double_indexed` \* *X*, `double_complex_indexed` \* *Y* )

Set indexed complex double precision to indexed double precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

#### 4.1.3.55 void `zidupdate` ( const int *fold*, const `double` *X*, `double_complex_indexed` \* *Y* )

Update indexed complex double precision with double precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.56 `void zinegate ( const int fold, double_complex_indexed * X )`

Negate indexed complex double precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.57 `int zinum ( const int fold )`

indexed complex double precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in `double`) of the indexed type

## Author

Peter Ahrens

## Date

27 Apr 2015

4.1.3.58 `void ziprint ( const int fold, const double_complex_indexed * X )`

Print indexed complex double precision.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.59 `void zirenorm ( const int fold, double_complex_indexed * X )`

Renormalize indexed complex double precision.

Renormalization keeps the rep vector within the necessary bounds by shifting over to the carry vector

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.60 void zisetzero ( const int *fold*, double\_complex\_indexed \* *X* )**

Set indexed double precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.61 size\_t zisize ( const int *fold* )**

indexed complex double precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in bytes) of the indexed type

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**4.1.3.62 void zizadd ( const int *fold*, const void \* *X*, double\_complex\_indexed \* *Y* )**

Add complex double precision to indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type Y

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.63 void zizconv ( const int *fold*, const void \* *X*, double\_complex\_indexed \* *Y* )

Convert complex double precision to indexed complex double precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.64 void zizdeposit ( const int *fold*, const void \* *X*, double\_complex\_indexed \* *Y* )

Add complex double precision to suitably indexed manually specified indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type Y where the index of Y is larger than the index of X

## Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [zizupdate\(\)](#) on Y with the maximum absolute value of any elements you wish to deposit in Y, you can call this method to deposit a maximum of [dicapacity\(\)](#) elements into Y. After calling [zizdeposit\(\)](#) on an indexed type, you must renormalize the indexed type with [zirenorm\(\)](#).

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>repY</i>	Y's rep vector

<i>incrpY</i>	stride within Y's rep vector (use every incrpY'th element)
---------------	--

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.65** void ziziadd ( const int *fold*, const double\_complex\_indexed \* X, double\_complex\_indexed \* Y )

Add indexed complex double precision (Y += X)

Performs the operation Y += X

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.66** void ziziset ( const int *fold*, const double\_complex\_indexed \* X, double\_complex\_indexed \* Y )

Set indexed complex double precision (Y = X)

Performs the operation Y = X

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**4.1.3.67** void zizupdate ( const int *fold*, const void \* X, double\_complex\_indexed \* Y )

Update indexed complex double precision with complex double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

4.1.3.68 `void zziconv_sub ( const int fold, const double_complex_indexed * X, void * conv )`

Convert indexed complex double precision to complex double precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>conv</i>	scalar return

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015





# Index

cciconv\_sub  
    indexed.h, [13](#)  
cialloc  
    indexed.h, [13](#)  
cicadd  
    indexed.h, [15](#)  
cicconv  
    indexed.h, [15](#)  
cicdeposit  
    indexed.h, [15](#)  
ciciadd  
    indexed.h, [16](#)  
ciciset  
    indexed.h, [16](#)  
cicupdate  
    indexed.h, [17](#)  
cinegate  
    indexed.h, [17](#)  
cinum  
    indexed.h, [17](#)  
ciprint  
    indexed.h, [18](#)  
cirenorm  
    indexed.h, [18](#)  
cisetzero  
    indexed.h, [18](#)  
cisiset  
    indexed.h, [19](#)  
cisize  
    indexed.h, [19](#)  
cisupdate  
    indexed.h, [19](#)  
  
dlComplex\_, [5](#)  
dbound  
    indexed.h, [20](#)  
ddiconv  
    indexed.h, [20](#)  
dialloc  
    indexed.h, [20](#)  
dicapacity  
    indexed.h, [21](#)  
didadd  
    indexed.h, [21](#)  
didconv  
    indexed.h, [21](#)  
diddeposit  
    indexed.h, [22](#)  
didiadd  
    indexed.h, [22](#)  
  
didiset  
    indexed.h, [23](#)  
didupdate  
    indexed.h, [23](#)  
dindex  
    indexed.h, [23](#)  
dinegate  
    indexed.h, [24](#)  
dinum  
    indexed.h, [24](#)  
diprint  
    indexed.h, [24](#)  
direnorm  
    indexed.h, [25](#)  
disetzero  
    indexed.h, [25](#)  
disize  
    indexed.h, [25](#)  
diwidth  
    indexed.h, [26](#)  
double\_complex\_indexed  
    indexed.h, [12](#)  
double\_indexed  
    indexed.h, [12](#)  
  
float\_complex\_indexed  
    indexed.h, [13](#)  
float\_indexed  
    indexed.h, [13](#)  
  
ldouble\_, [5](#)  
lfloat\_, [5](#)  
include/indexed.h, [7](#)  
indexed.h  
    cciconv\_sub, [13](#)  
    cialloc, [13](#)  
    cicadd, [15](#)  
    cicconv, [15](#)  
    cicdeposit, [15](#)  
    ciciadd, [16](#)  
    ciciset, [16](#)  
    cicupdate, [17](#)  
    cinegate, [17](#)  
    cinum, [17](#)  
    ciprint, [18](#)  
    cirenorm, [18](#)  
    cisetzero, [18](#)  
    cisiset, [19](#)  
    cisize, [19](#)  
    cisupdate, [19](#)

- dbound, 20
- ddiconv, 20
- dialloc, 20
- dicapacity, 21
- didadd, 21
- didconv, 21
- diddeposit, 22
- didiadd, 22
- didiset, 23
- didupdate, 23
- dindex, 23
- dinegate, 24
- dinum, 24
- diprint, 24
- direnorm, 25
- disetzero, 25
- disize, 25
- diwidth, 26
- double\_complex\_indexed, 12
- double\_indexed, 12
- float\_complex\_indexed, 13
- float\_indexed, 13
- sbound, 26
- sialloc, 26
- sicapacity, 27
- sindex, 27
- sinegate, 27
- sinum, 28
- siprint, 28
- sirenorm, 28
- sisadd, 30
- sisconv, 30
- sisdeposit, 30
- sisetzero, 31
- sisiadd, 31
- sisiset, 32
- sisize, 32
- sisupdate, 32
- siwidth, 34
- ssiconv, 34
- zialloc, 34
- zidiset, 35
- zidupdate, 35
- zinegate, 35
- zinum, 37
- ziprint, 37
- zirenorm, 37
- zisetzero, 38
- zisize, 38
- zizadd, 38
- zizconv, 39
- zizdeposit, 39
- ziziadd, 40
- ziziset, 40
- zizupdate, 40
- zziconv\_sub, 41
- indexed.h, 26
- sialloc
  - indexed.h, 26
- sicapacity
  - indexed.h, 27
- sindex
  - indexed.h, 27
- sinegate
  - indexed.h, 27
- sinum
  - indexed.h, 28
- siprint
  - indexed.h, 28
- sirenorm
  - indexed.h, 28
- sisadd
  - indexed.h, 30
- sisconv
  - indexed.h, 30
- sisdeposit
  - indexed.h, 30
- sisetzero
  - indexed.h, 31
- sisiadd
  - indexed.h, 31
- sisiset
  - indexed.h, 32
- sisize
  - indexed.h, 32
- sisupdate
  - indexed.h, 32
- siwidth
  - indexed.h, 34
- ssiconv
  - indexed.h, 34
- zialloc
  - indexed.h, 34
- zidiset
  - indexed.h, 35
- zidupdate
  - indexed.h, 35
- zinegate
  - indexed.h, 35
- zinum
  - indexed.h, 37
- ziprint
  - indexed.h, 37
- zirenorm
  - indexed.h, 37
- zisetzero
  - indexed.h, 38
- zisize
  - indexed.h, 38
- zizadd
  - indexed.h, 38
- zizconv
  - indexed.h, 39
- zizdeposit
- slComplex\_, 6
- sbound

---

indexed.h, [39](#)  
ziziadd  
indexed.h, [40](#)  
ziziset  
indexed.h, [40](#)  
zizupdate  
indexed.h, [40](#)  
zziconv\_sub  
indexed.h, [41](#)