

ReproBLAS

Generated by Doxygen 1.8.9.1

Thu May 21 2015 19:35:00



# Contents



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

|                                       |    |
|---------------------------------------|----|
| dIComplex_ . . . . .                  | ?? |
| Idouble_ . . . . .                    | ?? |
| Ifloat_ . . . . .                     | ?? |
| sIComplex_ . . . . .                  | ?? |
| Target                                |    |
| src.indexed.deposit.Deposit . . . . . | ?? |



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

|   |    |
|---|----|
| <a href="#">src.indexed.deposit.Deposit</a> | ?? |
| <a href="#">dlComplex_</a>                  | ?? |
| <a href="#">ldouble_</a>                    | ?? |
| <a href="#">lfloat_</a>                     | ?? |
| <a href="#">slComplex_</a>                  | ?? |





## Chapter 3

# File Index

### 3.1 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 . . . ??

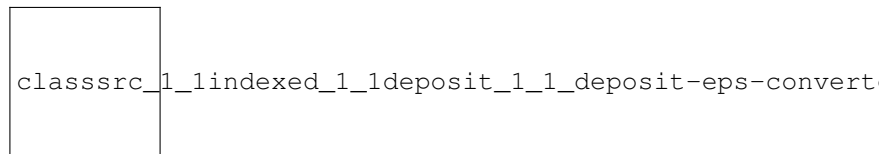


## Chapter 4

# Data Structure Documentation

### 4.1 src.indexed.deposit.Deposit Class Reference

Inheritance diagram for src.indexed.deposit.Deposit:



#### Public Member Functions

- def **\_\_init\_\_** (self, data\_type\_class, N, X, incX, manY, incmanY)
- def **get\_arguments** (self)
- def **get\_metrics** (self)
- def **get\_parameters** (self)
- def **write** (self, code\_block)
- def **write\_vec** (self, vec\_class, code\_block)
- def **write\_fold** (self, code\_block, fold, max\_pipe\_width, max\_unroll\_width)
- def **write\_increments** (self, code\_block, fold, max\_pipe\_width, max\_unroll\_width)
- def **write\_core** (self, code\_block, fold, max\_pipe\_width, max\_unroll\_width, incs)
- def **preprocess**
- def **process** (self, code\_block, fold, reg\_width, unroll\_width)
- def **define\_load\_vars** (self, code\_block, width)
- def **define\_load\_ptrs** (self, code\_block, width)
- def **compute\_reg\_width** (self, pipe\_width)

#### Data Fields

- **default\_fold**
- **max\_fold**
- **max\_expand\_fold**
- **data\_type\_class**
- **N**
- **X**
- **incX**
- **manY**

- **incmanY**
- **code\_block**
- **data\_type**
- **vec**
- **compression\_var**
- **q\_vars**
- **s\_vars**
- **buffer\_vars**
- **load\_vars**
- **load\_ptrs**

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

- [src/indexed/deposit.py](#)

## 4.2 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](#)

## 4.3 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](#)

## 4.4 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](#)

## 4.5 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 5

# File Documentation

### 5.1 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 dmexpansion` ()
- `double dmcompression` ()
- `double dierror` (const int fold, const int N, const double X)
- `double dbin` (const int X)  
*Get double precision bin corresponding to index.*
- `void dmbin` (const int fold, const int X, double \*manY, const int incmanY, double \*carY, const int inccarY)
- `int dindex` (const double X)  
*Get index of double precision.*
- `int dmindex` (const double \*manX)
- `float smexpansion` ()
- `float smcompression` ()
- `float sierror` (const int fold, const int N, const float X)
- `float sbin` (const int X)  
*Get single precision bin corresponding to index.*
- `void smbin` (const int fold, const int X, float \*manY, const int incmanY, float \*carY, const int inccarY)
- `int sindex` (const float X)  
*Get index of single precision.*
- `int smindex` (const float \*manX)



- void `ciprint` (const int fold, const `float_complex_indexed` \*X)  
*Print indexed complex single precision.*
- void `cmprint` (const int fold, const float \*manX, const int incmanX, const float \*carX, const int inccarX)
- void `diprint` (const int fold, const `double_indexed` \*X)  
*Print indexed double precision.*
- void `dmpprint` (const int fold, const double \*manX, const int incmanX, 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 \*manX, const int incmanX, 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 \*manX, const int incmanX, const double \*carX, const int inccarX)
- void `didisub` (const int fold, const `double_indexed` \*X, `double_indexed` \*Y)  
*Set indexed double precision ( $Y = X$ )*
- void `dmdmset` (const int fold, const double \*manX, const int incmanX, const double \*carX, const int inccarX, double \*manY, const int incmanY, 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 \*manX, const int incmanX, const double \*carX, const int inccarX, double \*manY, const int incmanY, 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 \*manX, const int incmanX, const double \*carX, const int inccarX, double \*manY, const int incmanY, double \*carY, const int inccarY)
- void `sisisub` (const int fold, const `float_indexed` \*X, `float_indexed` \*Y)  
*Set indexed single precision ( $Y = X$ )*
- void `smsmset` (const int fold, const float \*manX, const int incmanX, const float \*carX, const int inccarX, float \*manY, const int incmanY, 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 \*manX, const int incmanX, const float \*carX, const int inccarX, float \*manY, const int incmanY, 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 \*manX, const int incmanX, const float \*carX, const int inccarX, float \*manY, const int incmanY, 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 \*manX, const int incmanX, 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 \*manX, const int incmanX, 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 \*manX, const int incmanX, 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 \*manX, const int incmanX, 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 \*manX, const int incmanX, const double \*carX, const int inccarX, double \*manY, const int incmanY, 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 \*manX, const int incmanX, const double \*carX, const int inccarX, double \*manY, const int incmanY, 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 \*manX, const int incmanX, const float \*carX, const int inccarX, float \*manY, const int incmanY, 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 \*manX, const int incmanX, const float \*carX, const int inccarX, float \*manY, const int incmanY, 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 \*manY, const int incmanY, 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 \*manY, const int incmanY, 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 \*manY, const int incmanY, 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 \*manY, const int incmanY, float \*carY, const int inccarY)
- void **didupdate** (const int fold, const double X, double\_indexed \*Y)

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

- void **dmdupdate** (const int fold, const double X, double \*manY, const int incmanY, 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 ( $X \rightarrow Y$ )*

- void **zmzupdate** (const int fold, const void \*X, double \*manY, const int incmanY, 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 ( $X \rightarrow Y$ )*

- void **zmdupdate** (const int fold, const double X, double \*manY, const int incmanY, double \*carY, const int inccarY)
- void **sisupdate** (const int fold, const float X, float\_indexed \*Y)

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

- void **smsupdate** (const int fold, const float X, float \*manY, const int incmanY, 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 ( $X \rightarrow Y$ )*

- void **cmcupdate** (const int fold, const void \*X, float \*manY, const int incmanY, 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 ( $X \rightarrow Y$ )*

- void **cmsupdate** (const int fold, const float X, float \*manY, const int incmanY, 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 \*manY, const int incmanY)
- 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 \*manY, const int incmanY)
- 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 \*manY, const int incmanY)
- 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 \*manY, const int incmanY)
- void **direnorm** (const int fold, [double\\_indexed](#) \*X)
 

*Renormalize indexed double precision.*
- void **dmrenorm** (const int fold, double \*manX, const int incmanX, 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 \*manX, const int incmanX, double \*carX, const int inccarX)
- void **sirenorm** (const int fold, [float\\_indexed](#) \*X)
 

*Renormalize indexed single precision.*
- void **smrenorm** (const int fold, float \*manX, const int incmanX, 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 \*manX, const int incmanX, float \*carX, const int inccarX)
- void **didconv** (const int fold, const double X, [double\\_indexed](#) \*Y)
 

*Convert double precision to indexed double precision ( $X \rightarrow Y$ )*
- void **dmdconv** (const int fold, const double X, double \*manY, const int incmanY, 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 ( $X \rightarrow Y$ )*
- void **zmzconv** (const int fold, const void \*X, double \*manY, const int incmanY, double \*carY, const int inccarY)
- void **sisconv** (const int fold, const float X, [float\\_indexed](#) \*Y)
 

*Convert single precision to indexed single precision ( $X \rightarrow Y$ )*
- void **smsconv** (const int fold, const float X, float \*manY, const int incmanY, 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 ( $X \rightarrow Y$ )*
- void **cmcconv** (const int fold, const void \*X, float \*manY, const int incmanY, float \*carY, const int inccarY)
- double **ddiconv** (const int fold, const [double\\_indexed](#) \*X)
 

*Convert indexed double precision to double precision ( $X \rightarrow Y$ )*
- double **ddmconv** (const int fold, const double \*manX, const int incmanX, 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 ( $X \rightarrow Y$ )*
- void **zzmconv\_sub** (const int fold, const double \*manX, const int incmanX, const double \*carX, const int inccarX, void \*conv)
- float **ssiconv** (const int fold, const [float\\_indexed](#) \*X)
 

*Convert indexed single precision to single precision ( $X \rightarrow Y$ )*
- float **ssmconv** (const int fold, const float \*manX, const int incmanX, 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 ( $X \rightarrow Y$ )*
- void **ccmconv\_sub** (const int fold, const float \*manX, const int incmanX, 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 \*manX, const int incmanX, 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 \*manX, const int incmanX, 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 \*manX, const int incmanX, float \*carX, const int inccarX)
- void **cinigate** (const int fold, [float\\_complex\\_indexed](#) \*X)

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

- void **cmnegate** (const int fold, float \*manX, const int incmanX, float \*carX, const int inccarX)
- double **ufp** (const double X)
- float **ufpf** (const float X)

### 5.1.1 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 `dfop` would perform the function `fop` 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.

## 5.1.2 Typedef Documentation

### 5.1.2.1 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()`

### 5.1.2.2 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()`

### 5.1.2.3 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()`

### 5.1.2.4 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()`

## 5.1.3 Function Documentation

### 5.1.3.1 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

5.1.3.2 `float_complex_indexed* cialloc ( 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

5.1.3.3 `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 X                      |
| <i>Y</i>    | indexed scalar Y              |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.4 `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 X                      |
| <i>Y</i>    | indexed scalar Y              |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.5 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>manY</i>    | <i>Y</i> 's mantissa vector  |
| <i>incmanY</i> | stride within <i>Y</i> 's mantissa vector (use every <i>incmanY</i> 'th element) |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.6 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 |
|-------------|-------------------------------|

|     |                    |
|-----|--------------------|
| $X$ | indexed scalar $X$ |
| $Y$ | indexed scalar $Y$ |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

### 5.1.3.7 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 |
| $X$         | indexed scalar $X$            |
| $Y$         | indexed scalar $Y$            |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

### 5.1.3.8 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 |
| $X$         | scalar $X$                    |
| $Y$         | indexed scalar $Y$            |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

### 5.1.3.9 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 <i>X</i>       |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.10 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

5.1.3.11 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

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

Renormalize indexed complex single precision.

Renormalization keeps the mantissa vector within the necessary bins 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

5.1.3.13 `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

5.1.3.14 `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

5.1.3.15 `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

5.1.3.16 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 <i>X</i>               |
| <i>Y</i>    | indexed scalar <i>Y</i>       |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.17 double dbin ( const int *X* )

Get double precision bin corresponding to index.

## Parameters

|          |       |
|----------|-------|
| <i>X</i> | index |
|----------|-------|

## Returns

bin

## Author

Peter Ahrens

## Date

27 Apr 2015

5.1.3.18 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

**5.1.3.19 `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

**5.1.3.20 `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

5.1.3.21 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

5.1.3.22 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

5.1.3.23 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

**5.1.3.24 void ddiadd ( 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 <i>X</i>       |
| <i>Y</i>    | indexed scalar <i>Y</i>       |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**5.1.3.25 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 |
| <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

**5.1.3.26 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 |
| <i>X</i>    | scalar X                      |
| <i>Y</i>    | indexed scalar Y              |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**5.1.3.27 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 X |
|----------|----------|

**Returns**

X's index

**Author**

Peter Ahrens  
Hong Diep Nguyen

**Date**

19 May 2015

**5.1.3.28 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 X              |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015



5.1.3.29 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

**5.1.3.30 void dprint ( 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 <i>X</i>       |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**5.1.3.31 void direnorm ( const int *fold*, `double_indexed` \* *X* )**

Renormalize indexed double precision.

Renormalization keeps the mantissa vector within the necessary bins 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

**5.1.3.32 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 <i>X</i>       |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.33 `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

5.1.3.34 `int diwidth ( )`

Get indexed double precision bin width.

## Returns

bin width (in bits)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.35 `float sbin ( const int X )`

Get single precision bin corresponding to index.

**Parameters**

|          |       |
|----------|-------|
| <i>X</i> | index |
|----------|-------|

**Returns**

bin (bin)

**Author**

Peter Ahrens

**Date**

27 Apr 2015

**5.1.3.36 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

**5.1.3.37 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

#### 5.1.3.38 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 indicies correspond to smaller bins.

## Parameters

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

## Returns

*X*'s index

## Author

Peter Ahrens  
Hong Diep Nguyen

## Date

19 May 2015

5.1.3.39 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

5.1.3.40 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

5.1.3.41 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 X              |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

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

Renormalize indexed single precision.

Renormalization keeps the mantissa vector within the necessary bins 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

5.1.3.43 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

5.1.3.44 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

5.1.3.45 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

5.1.3.46 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

**5.1.3.47 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

**5.1.3.48 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

**5.1.3.49 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

**5.1.3.50** `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 <i>X</i>               |
| <i>Y</i>    | indexed scalar <i>Y</i>       |

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**5.1.3.51** `int siwidth ( )`

Get indexed single precision bin width.

**Returns**

bin width (in bits)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**5.1.3.52** `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

**5.1.3.53 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

**5.1.3.54 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 X              |
| <i>Y</i>    | indexed scalar Y              |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

### 5.1.3.55 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

### 5.1.3.56 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

### 5.1.3.57 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

5.1.3.58 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 X              |

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

5.1.3.59 void zirenorm ( const int *fold*, double\_complex\_indexed \* *X* )

Renormalize indexed complex double precision.

Renormalization keeps the mantissa vector within the necessary bins 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

5.1.3.60 void zisetzzero ( 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

5.1.3.61 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

**5.1.3.62 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

**5.1.3.63 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

#### 5.1.3.64 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>manY</i>    | $Y$ 's mantissa vector   |
| <i>incmanY</i> | stride within $Y$ 's mantissa vector (use every $incmanY$ 'th element) |

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

#### 5.1.3.65 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

#### 5.1.3.66 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

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

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

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

**5.1.3.68 void zzconv\_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

