Simple and Light Interfaces for C and C++ users

SLLIB – Script-Like C-language library
User's Reference Guide Advanced Edition

Version 1.1.0 English

Credits

Software Development:

Chisato Yamauchi

Quality Assurance:

SEC Co., LTD.

Manual Document:

Chisato Yamauchi, *Sachimi Fujishima* and *SEC Co., LTD.*

Manual Translation:

Space Engineering Development Co.,LTD. and *Sakura Academia Corporation*

Special Thanks:

Daisuke Ishihara, *Hajime Baba*, *Iku Shinohara*, *Keiichi Matsuzaki*, *Sergio Pascual* and *Yukio Yamamoto*

Web page: `http://www.ir.isas.jaxa.jp/~cyamauch/sli/`

Contents

1 TARRAY Template Class

The tarray template class can handle arrays of arbitrary data types or classes.

Its function is almost the same as that of tarray_tstring in the Basic Edition, except that this class does not include member functions specializing in string. Therefore, the EXAMPLE section is omitted in this chapter. As for EXAMPLE, see the references of each member function of tarray_tstring in the Basic Edition.

To use the tarray template class, write the following code followed by the user code.

```
#include <sli/tarray.h>
```

Additionally, write "`using namespace sli;`" in the code if the namespace declaration is required.

1.1 How to Create an Object

To create an object of the tarray template class, no arguments are required. Create it normally as shown below:

```
#include <sli/tarray.h>
using namespace sli;

int main()
{
    tarray<tarray_tstring> my_2d_array;
```

In this example, the array of tarray_tstring in the Basic Edition is created. This make it easy to handle a two dimensional array of string as shown below.

```
    my_2d_array[0][0] = "SLLIB";
```

1.2 List of Member Functions

The member functions are listed in Table 1.

	Function Name	Description
§1.3.1	`[]`	A reference to the specified element
§1.3.2	`=`	Substitute an array for the same type and class
§1.3.3	`+=`	Append an array
§1.3.4	`+=`	Append an element
§1.4.1	`length()`	Length of the array (the number of elements)
§1.4.2	`at()`, `at_cs()`	A reference to the specified element
§1.4.3	`copy()`	Copy (a part of) the array itself to another one
§1.4.4	`swap()`	Replace the array itself with that of the external object
§1.4.5	`init()`	Initialize the object completely
§1.4.6	`assign()`	Initialize and substitute the object
§1.4.7	`put()`	Put a value to an arbitrary element's point
§1.4.8	`append()`	Append an element
§1.4.9	`insert()`	Insert an element
§1.4.10	`replace()`	Replace an element
§1.4.11	`erase()`	Erase an element
§1.4.12	`clean()`	Padding of the existing values in an array by arbitrary ones
§1.4.13	`resize()`	Change the length of the array
§1.4.14	`resizeby()`	Change the length of the array relatively
§1.4.15	`crop()`	Extract an array

Table 1: List of Member Functions Available in tarray Template Class.

1.3 Operators

1.3.1 []

NAME

`[]` — A reference to the specified element

SYNOPSIS

```
type &operator[]( size_t index );  ..... 1
const type &operator[]( size_t index ) const;  ..... 2
```

DESCRIPTION

This operator returns a reference to an array element specified by the index. The member functions in the `type` class can be called by specifying them after this operator and the "." operator.

The member function 1 is available for read/write and corresponds to `at()`. The member function 2 is available for read only and corresponds to `at_cs()`.

When the specified `index` exceeds the length of the array, the member function 1 extends the array size automatically and the member function 2 throws an exception. The initial element index of the array is `0`.

Whether the member function 1 or 2 is used depend on whether the object has the "const " attribute. The member function 1 is used for the object without the "const " attribute, and the function 2 is used with the attribute automatically.

For more information about `at()`, `at_cs()`, see §1.4.2.

PARAMETER

[I] `index` Array subscript that starts from `0`

RETURN VALUE

A reference to the element specified by the index in the array

EXCEPTION

- The member function 1 throws an exception when it fails to allocate a local buffer.
- The function 2 throws an exception when the argument `index` is greater than or equal to the number of elements in this array.

1.3.2 =

NAME

= — Substitute an array for the same type and class

SYNOPSIS

```
tarray &operator=(const tarray &obj);
```

DESCRIPTION

This operator substitutes the array with the same data type or class as that of the object itself specified by the right side of the operator (argument) into the object itself.

PARAMETER

[I] `obj` An object containing an array of elements of the same type as those on the left side

RETURN VALUE

A reference to itself

EXCEPTION
The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

1.3.3 +=

NAME
+= — Append an array

SYNOPSIS
```
tarray &operator+=(const tarray &obj);
```

DESCRIPTION
This operator appends the array specified by the right side of the operator (argument) to the object itself.

PARAMETER
[I] `obj` An object containing an array of elements of the same type as those on the left side

RETURN VALUE
A reference to itself

EXCEPTION
The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

1.3.4 +=

NAME
+= — Append an element

SYNOPSIS
```
tarray &operator+=(const type &one);
```

DESCRIPTION
This operator appends one element specified by the right side of the operator (argument) to the object itself.

PARAMETER
[I] `one` An element to be appended to this array

RETURN VALUE
A reference to itself

EXCEPTION
The function throws an exception when it fails to allocate a local buffer.

1.4 The member functions

1.4.1 length()

NAME

length() — Length of the array (the number of elements)

SYNOPSIS

```
size_t length() const;
```

DESCRIPTION

`length()` returns length the (number of elements) of an array.

RETURN VALUE

Length of an array

1.4.2 at(), at_cs()

NAME

at(), at_cs() — A reference to the specified element

SYNOPSIS

```
type &at( size_t index );  ..... 1
const type &at( size_t index ); const  ..... 2
const type &at_cs( size_t index ) const;  ..... 3
```

DESCRIPTION

`at()` and `at_cs()` return a reference to the element of an array specified by `index`. The member functions in the `type` class can be used by placing the "`.`" operator immediately after these member functions.

The member function 1 can be used for both read/write. while the member functions 2 and 3 are used for read only.

As for the `at()` member function, whether the member function 1 or 2 is used depends on whether the object has the `const` attribute. Without the `const`, the member function 1 is selected. With the `const` attribute, the member function 2 is selected automatically.

If a value equal to or greater than the length of the array is given to `index` for the member function 1, a new element of array is created. No exception is thrown except when it fails to allocate a buffer.

If a value equal to or greater than the length of the array is given to `index` for the member function 2 or 3, an exception is thrown.

PARAMETER

[I] `index` Subscript specifying an array element
([I] : input, [O] : output)

RETURN VALUE

A reference to the value or the object corresponding to the specified element's index

EXCEPTION

- The member function 1 throws an exception when it fails to allocate a local buffer. - The member functions 2 and 3 throw an exception when the argument `index` is greater than or equal to the number of elements in this array.

1.4.3 copy()

NAME

copy() — Copy (a part of) the array itself to another one

SYNOPSIS

```
ssize_t copy( tarray *dest ) const;  ..... 1
ssize_t copy( size_t index, tarray *dest ) const;  ..... 2
ssize_t copy( size_t index, size_t n, tarray *dest ) const;  ..... 3
ssize_t copy( tarray &dest ) const;  ..... 4
ssize_t copy( size_t index, tarray &dest ) const;  ..... 5
ssize_t copy( size_t index, size_t n, tarray &dest ) const;  ..... 6
```

DESCRIPTION

`copy()` copies all or a part of its own array to the object specified by `dest`. The return value is the number of elements written to `dest`.

The argument `dest` is available in the forms of a pointer variable for member function 1-3 and a reference for member functions 4-6. These behave exactly the same. A user's policy should specify which one is to be used.

The member functions 1 and 4 copy all elements in an array to `dest`.

The member functions 2, 3, 5 and 6 copy elements starting from the index of the element in an array, which is `index`. The index at the beginning of the array is `0`. As for the member functions 3 and 6, the number of elements to be copied, `n`, can be specified.

If the value of
index + `n` exceeds the number of elements in the source array, the elements from `index` to the last are copied. If the value of `index` exceeds the number of elements in the source array, the content of `dest` is deleted and they return `-1`.

PARAMETER

[O]	`dest`	A pointer to an instance of the class "tarray" to which a subarray of this object should be written
[I]	`index`	Subscript specifying the first element of the subarray
[I]	`n`	Number of elements in the subarray

([I] : input, [O] : output)

RETURN VALUE

Non-negative value	:	Number of elements copied
Negative value (error)	:	If a value equal to or greater than length of the array is specified to `index`

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.4 swap()

NAME

swap() — Replace the array itself with that of the external object

SYNOPSIS

```
tarray &swap( tarray &sobj );
```

DESCRIPTION

`swap()` swaps the content of the caller with that of the object `sobj`.

PARAMETER

[I/O] `sobj` An instance of the class "tarray" whose contents should be swapped with those in this object

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

1.4.5 init()

NAME

init() — Initialize the object completely

SYNOPSIS

```
tarray &init();  ..... 1
tarray &init(const tarray &obj);  ..... 2
```

DESCRIPTION

`init()` initializes the caller's array.

The member function 1 initializes an object completely, and a memory area allocated to an array buffer is released completely.

The member function 2 initializes with the content of the specified object `obj` (copies the whole content of `obj` to the caller object).

PARAMETER

[I] `obj` An instance of the class "tarray" whose contents should be copied to this object as its initial value

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

Both functions throw an exception when they fail to allocate a local buffer. In addition, the member function 2 also throws an exception when it detects memory corruption.

1.4.6 assign()

NAME

assign() — Initialize and substitute the object

SYNOPSIS

```
tarray &assign( const type &one, size_t n );  ..... 1
tarray &assign( const tarray &src, size_t idx2 = 0 );  ..... 2
tarray &assign( const tarray &src, size_t idx2, size_t n2 );  ..... 3
```

DESCRIPTION

The member function 1 initializes the caller's array with `n` number of elements that have a value specified by `one`.

The member functions 2 and 3 assign `n2` number of elements starting from the element's index `idx2` in the array `src` to the caller's array.

PARAMETER

[I] `one` A reference to a scalar with which this array is filled
[I] `n` Number of entries to which the scalar `one` is written
[I] `src` A reference to the object that contains an input array
[I] `idx2` Array subscript specifying the first element to be extracted from the input array `src`
[I] `n2` Number of elements to be extracted from the input array `src`

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.7 put()

NAME

put() — Put a value to an arbitrary element's point

SYNOPSIS

```
tarray &put( size_t index, const type &one, size_t n );  ..................  1
tarray &put( size_t index, const tarray &src, size_t idx2 = 0 );  .........  2
tarray &put( size_t index, const tarray &src, size_t idx2, size_t n2 );  .  3
```

DESCRIPTION

The member function 1 overwrites the caller's array with `n` number of elements that have a value specified by `one` starting from the position of the element's index `index`.

The member functions 2 and 3 overwrite the caller's array with `n2` number of elements starting from the element's index `idx2` in the array `src` starting from the position of the element's index `index`.

The element's index at the beginning of the array is zero.

Arbitrary values can be given to `index`. If the specified argument exceeds the number of elements in an array, the size of the array is extended automatically. When the array is empty, for example, the results of `my_arr.put(0,value,6)` and `my_arr.put(2,value,4)` are the same. When the array has four elements, `my_arr.put(2,value,4)` makes the number of elements in the array six. The elements after the element's index 2 are the ones specified by `value`.

PARAMETER

[I] `index` Subscript specifying the first element of a subarray of this object to which the given data should be written
[I] `one` A reference to a scalar to be written to the subarray
[I] `n` Number of entries to which the scalar `one` is written
[I] `src` A reference to an object that contains an input array
[I] `idx2` Array subscript specifying the first element to be extracted from the input array `src`
[I] `n2` Number of elements to be extracted from the input array `src`

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.8 append()

NAME

append() — Append an element

SYNOPSIS

```
tarray &append( const type &one, size_t n );  ..............................  1
tarray &append( const tarray &src, size_t idx2 = 0 );  ......................  2
tarray &append( const tarray &src, size_t idx2, size_t n2 );  ..............  3
```

DESCRIPTION

The member function 1 appends `n` number of elements that have a value specified by `one` to the end of the caller's array.

The member functions 2 and 3 append `n2` number of elements starting from the element's index `idx2` in the array `src` to the end of the caller's array.

The element's index at the beginning of the array is zero.

PARAMETER

[I]	`one`	A reference to a scalar to be written to an additional subarray of this object
[I]	`n`	Number of entries to which the scalar `one` is written
[I]	`src`	A reference to an object containing an input array
[I]	`idx2`	Array subscript specifying the first element to be extracted from the input array `src`
[I]	`n2`	Number of elements to be extracted from the input array `src`

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.9 insert()

NAME

insert() — Insert an element

SYNOPSIS

```
tarray &insert( size_t index, const type &one, size_t n );  ................  1
tarray &insert( size_t index,
                const tarray &src, size_t idx2 = 0 );  ......................  2
tarray &insert( size_t index,
                const tarray &src, size_t idx2, size_t n2 );  ..............  3
```

DESCRIPTION

The member function 1 inserts `n` number of elements that have a value specified by `one` to the position of the element's index `index` in the caller's array.

The member functions 2 and 3 insert `n2` number of elements starting from the element's index `idx2` in the array `src` to the position of the element's index `index` in the caller's array.

The element's index at the beginning of the array is zero.

If a value equal to or greater than the length of the array is given to `index`, it is assumed that the length of the array is given to `index`.

PARAMETER

[I]	`index`	Array subscript specifying an element before which a new additional array should be inserted
[I]	`one`	A reference to a scalar to be written to the additional subarray of this object
[I]	`n`	Number of new entries to which the scalar `one` is written
[I]	`src`	A reference to an object containing an input array
[I]	`idx2`	Array subscript specifying the first element to be extracted from the input array `src`
[I]	`n2`	Number of elements to be extracted from the input array `src`

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

1.4.10 replace()

NAME

replace() — Replace an element

SYNOPSIS

```
tarray &replace( size_t idx1, size_t n1,
                 const type &one, size_t n );  ............................. 1
tarray &replace( size_t idx1, size_t n1,
                 const tarray &src, size_t idx2 = 0 );  ................... 2
tarray &replace( size_t idx1, size_t n1,
                 const tarray &src, size_t idx2, size_t n2 );  ............ 3
```

DESCRIPTION

The member function 1 replaces `n1` number of elements starting from the element's index `idx1` in the caller's array with `n` number of elements that have a value specified by `one`.

The member function 2 replaces `n1` number of elements starting from the element's index `idx1` in the caller's array with `n2` number of elements starting from the element's index `idx2` in the array `src`.

The element's index at the beginning of the array is zero.

If a value equal to or greater than the length of the array is given to `idx1`, it performs a similar processing to that of the `append()` member function (§1.4.8). When the sum of `idx1` and `n1` is greater than the number of elements in the array, or the extension or contraction of the array is required depending on the magnitude relation between `n1` and `n2`, it adjusts the number of elements automatically.

PARAMETER

[I]	`idx1`	Subscript specifying the first element of a subarray in this object
[I]	`n1`	Number of elements in the subarray to be replaced
[I]	`one`	A reference to a scalar that fills a new array of length `n`
[I]	`n`	Number of elements in the new array that replaces the original subarray in this object
[I]	`src`	A reference to an object containing an input array
[I]	`idx2`	Array subscript specifying the first element to be extracted from the input array `src`
[I]	`n2`	Number of elements to be extracted from the input array `src`

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.11 erase()

NAME

erase() — Erase an element

SYNOPSIS

```
tarray &erase();  ..................................................................  1
tarray &erase( size_t index, size_t num_elements = 1 );  ...................  2
```

DESCRIPTION

The member function 1 erases all elements in the caller's array. (The length of the array becomes 0.)

The member function 2 erases `num_elements` number of elements starting from the element's index `index`.

The element's index at the beginning of the array is zero. If `num_elements` is not specified, it erases one element.

If a value equal to or greater than the length of the array is given to `index`, it is ignored.

PARAMETER

[I]	`index`	Subscript specifying the first element of a subarray to be removed
[I]	`num_elements`	Number of elements in the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.12 clean()

NAME

clean() — Padding of the existing values in an array by arbitrary ones

SYNOPSIS

```
tarray &clean(const type &one);
```

DESCRIPTION

`clean()` performs the padding of all elements in the caller's array with the value `one`. The length of the array does not change even after `clean()` is executed.

PARAMETER

[I] `one` A reference to a scalar that should be written to all entries of this array
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer for elements.

1.4.13 resize()

NAME

resize() — Change the length of the array

SYNOPSIS

```
tarray &resize( size_t new_num_elements );
```

DESCRIPTION

`resize()` changes the length of the caller's array to `new_num_elements`. When the length of the array is contracted, the elements after `new_num_elements` are deleted.

PARAMETER

[I] `new_num_elements` Number of elements of this array after being resized
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.14 resizeby()

NAME

resizeby() — Change the length of the array relatively

SYNOPSIS

```
tarray &resizeby( ssize_t len );
```

DESCRIPTION

`resizeby()` changes the length of the caller's array by adding or subtracting the length of `len`.

When the length of the array is contracted, the last abs(`len`) number of elements are deleted.

PARAMETER

[I] `len` Number of elements to be appended to or removed from this array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

1.4.15 crop()

NAME

crop() — Extract an array

SYNOPSIS

```
tarray &crop( size_t idx, size_t len );
tarray &crop( size_t idx );
```

DESCRIPTION

`crop()` crops a part of the caller's array by leaving only `len` number of elements starting from the element's index `idx`. If `len` is omitted, it crops everything but the elements after `idx`.

PARAMETER

[I] `idx` Subscript specifying the first element of a subarray to be extracted from this object

[I] `len` Number of elements in the extracted subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

---

2 ASARRAY Template Class

The asarray template class can handle the associative arrays of arbitrary data types or classes.

Its function is almost the same as that of asarray_tstring in the Basic Edition, except that this class does not include member functions specializing in string. Therefore, the EXAMPLE section is omitted in this chapter. As for EXAMPLE, see the references of each member function of asarray_tstring in Basic Edition.

To use the asarray template class, write the following code followed by the user code:

```
#include <sli/asarray.h>
```

Additionally, write "`using namespace sli;`" in the code if the namespace declaration is required.

2.1 How to Create an Object

To create an object of the asarray template class, no arguments are required. Create it normally as shown below:

```
#include <sli/asarray.h>
using namespace sli;

int main()
{
    asarray<asarray_tstring> my_2d_aarray;
```

In this example, the array of asarray_tstring in the Basic Edition is created. This make it easy to handle a two dimensional array of string as shown below.

```
    my_2d_aarray["CONFIG"]["OS"] = "Linux";
```

2.2 List of Member Functions

Member functions are listed in Table 2.

	Function Name	Description
§2.3.1	`[]`	A reference to an element corresponding to the specified key
§2.3.2	`=`	Copy an object of the asarray class
§2.4.1	`length()`	Length of an associated array (the number of elements)
§2.4.2	`at()`, `atf()`	A reference to the element value corresponding to specified key or element number
§2.4.3	`at_cs()`, `atf_cs()`	A reference to the element value corresponding to specified key or element number(read only)
§2.4.4	`index()`	Acquire the element number corresponding to the key string
§2.4.5	`key()`	Acquire the key string corresponding to the element number
§2.4.6	`keys()`	A reference to the array object of the key string (read only)
§2.4.7	`values()`	A reference to the array object of the value (read only)
§2.4.8	`swap()`	Replace objects
§2.4.9	`init()`	Initialize an object completely
§2.4.10	`assign()`	Initialize an object and substitute an element
§2.4.11	`assign_keys()`	Assign multiple strings or arrays of strings to the key
§2.4.12	`split_keys()`	Split a string and assign to the key
§2.4.13	`append()`	Append an element
§2.4.14	`insert()`	Insert an element
§2.4.15	`erase()`	Erase an element (a set of a key and a value)
§2.4.16	`clean()`	Padding of the total existing values in an associated array by arbitrary ones
§2.4.17	`rename_a_key()`	Change a key string

Table 2: List of Member Functions Available in asarray Template Class

2.3 Operators

2.3.1 []

NAME

`[]` — A reference to an element corresponding to the specified key

SYNOPSIS

```
type &operator[]( const char *key );  ..... 1
const type &operator[]( const char *key ) const;  ..... 2
```

DESCRIPTION

This operator returns a reference to an element value in the associative array with which `key` is associated. Member functions in the `type` class can be called by specifying them after this operator and the "`.`" operator.

The member function 1 is available for read/write and corresponds to `at()`. The member function 2 is available for read only and corresponds to `at_cs()`.

When the key string not defined in the associative array is specified, the member function 1 appends the set of the key string and the default value (depending on the `type` class) to the array, and the member function 2 throws an exception.

Whether the member function 1 or 2 is used depends on whether the object has the `const` attribute. The member function 1 is used for the object without the `const` attribute, and the function 2 is used with the attribute automatically.

For more information about `at()`, `at_cs()`, see §2.4.2.

PARAMETER

[I] `key` A key string of the associative array

RETURN VALUE

A reference to the value of the element corresponding to the key in the associative array

EXCEPTION

- Both functions throw an exception when "NULL" is given as the argument `key`.
- The member function 1 throws an exception when it fails to allocate a local buffer.
- The member function 2 throws an exception when the specified key string is not found.

2.3.2 =

NAME

= — Copy an object of the asarray class

SYNOPSIS

```
asarray &operator=(const asarray &obj);
```

DESCRIPTION

This operator substitutes the `asarray` class object specified by the right side of the operator (argument) into the object itself.

PARAMETER

[I] `obj` A reference to an object in the class "asarray" whose contents should be copied to this array

RETURN VALUE
A reference to itself

EXCEPTION
The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

2.4 Member Functions

2.4.1 length()

NAME
length() — Length of an associated array (the number of elements)

SYNOPSIS
```
size_t length() const;
```

DESCRIPTION
`length()` returns the length (number of elements) of the caller's associative array.

RETURN VALUE
Length of a associative array

2.4.2 at(), atf()

NAME
at(), atf() — A reference to the element value corresponding to the specified key or element number

SYNOPSIS
```
type &at( const char *key );  ..... 1
type &atf( const char *fmt, ... );  ..... 2
type &vatf( const char *fmt, va_list ap );  ..... 3
type &at( size_t index );  ..... 4
const type &at( const char *key ) const;  ..... 5
const type &atf( const char *fmt, ... ) const;  ..... 6
const type &vatf( const char *fmt, va_list ap ) const;  ..... 7
const type &at( size_t index ) const;  ..... 8
```

DESCRIPTION
`at()` and `atf()` return a reference to an element value in the associative array corresponding to the key string (member functions 1-3, 5-7) or the element's index (member functions 4 and 8) of the argument.

Member functions in the `type` class can be used by placing the "`.`" operator immediately after these member functions. The member functions 1-4 can be used for both read/write, while the member functions 5-8 are read only.

As for the member functions 2, 3, 6 and 7, a key string to be specified can be set in the same format and variable argument as the `printf()` function. In the member functions 2 and 6, each elemental data in the variable length argument is converted depending on the conversion specification of `fmt`. In the member functions 3 and 7, the list of the variable length argument `ap` is converted depending on the conversion specification of `fmt`. See the manual of the `printf()` function in libc for `fmt`.

When a given key string is not found, a combination of the specified key string and default value (depending on the type class) is added to the associative array in the member functions 1-3, while an exception is thrown in the member functions 5-7.

If a value equal to or greater than the length of the array is given to `index` for the member function 4 or 8, an exception is thrown. The element's index at the beginning of the array is zero.

Whether the member functions 1-4 or 5-8 are used depends on whether the object has the `const` attribute. Without the `const`, member functions 1-4 are selected. With the attribute, the member functions 5-8 are selected automatically.

PARAMETER

[I]	`key`	A key string for the associative array
[I]	`fmt`	A format specification for defining a key string
[I]	`...`	A variable number of arguments that should be treated according to the format specification `fmt`
[I]	`ap`	A list of arguments of the `va_list` type that should be treated according to the format specification `fmt`
[I]	`index`	Array subscript of the integer type `size_t`

([I] : input, [O] : output)

RETURN VALUE

A reference to the value of the element corresponding to the specified key or element's index

EXCEPTION

- All overloaded functions throw an exception when "NULL" is given as the argument `key`.
- When a given key string is not found, the member functions 5, 6, and 7 throw an exception.
- The member functions 4 and 8 throw an exception when the specified integer subscript `index` is invalid.
- All overloaded functions except 4, 5, and 8 throw an exception when they fail to allocate a local buffer.

2.4.3 at_cs(), atf_cs()

NAME

at_cs(), atf_cs() — A reference to the element value corresponding to the specified key or element number (read only)

SYNOPSIS

```
const type &at_cs( const char *key ) const;  ..... 1
const type &atf_cs( const char *fmt, ... ) const;  ..... 2
const type &vatf_cs( const char *fmt, va_list ap ) const;  ..... 3
const type &at_cs( size_t index ) const;  ..... 4
```

DESCRIPTION

`at_cs()` and `atf_cs()` return a reference to an element value in the associative array corresponding to the key string (member function 1-3) or element's index (member function 4) of the argument. These member functions are for read only.

As for the member functions 2 and 3, a key string to be specified can be set in the same format and variable argument as the `printf()` function. In the member function 2, each elemental data in the variable length argument is converted depending on the conversion

specification of `fmt`. In the member function 3, the list of the variable length argument `ap` is converted depending on the conversion specification of `fmt`. See the manual of the `printf()` function in libc for `fmt`.

The element's index at the beginning of the array is zero.

When a given key string or element's index is not found, an exception is thrown.

PARAMETER

[I]	`key`	A key string for the associative array
[I]	`fmt`	A format specification for defining a key string
[I]	`...`	A variable number of arguments that should be treated according to the format specification `fmt`
[I]	`ap`	A list of arguments of the `va_list` type that should be treated according to the format specification `fmt`
[I]	`index`	Array subscript of the integer type `size_t`

([I] : input, [O] : output)

RETURN VALUE

A reference to the value of the element corresponding to the specified key or element's index

EXCEPTION

- All overloaded functions throw an exception when "NULL" is given as the argument `key`.
- When a given key string is not found, the member functions 1, 2, and 3 throw an exception.
- The member function 4 throws an exception when the specified integer subscript `index` is invalid.
- The member functions 2 and 3 throw an exception when they fail to allocate a local buffer.

2.4.4 index(), indexf(), vindexf()

NAME

index(), indexf(), vindexf() — Acquire the element number corresponding to the key string

SYNOPSIS

```
ssize_t index( const char *key ) const;  ..... 1
ssize_t indexf( const char *fmt, ... ) const;  ..... 2
size_t vindexf( const char *fmt, va_list ap ) const;  ..... 3
```

DESCRIPTION

`index()`, `indexf()` and `vindexf()` get the element's index corresponding to the key string. The element's index at the beginning of the array is zero.

As for the member function 2 and 3, a key string to be specified can be set in the same format and variable argument as the `printf()` function. In the member function 2, each elemental data in the variable length argument is converted depending on the conversion specification of `fmt`. In the member function 3, the list of the variable length argument `ap` is converted depending on the conversion specification of `fmt`. See the manual of the `printf()` function in libc for `fmt`.

PARAMETER

[I] `key` A key string for the associative array
[I] `fmt` A format specification for defining a key string
[I] `...` A variable number of arguments that should be treated according to the format specification `fmt`
[I] `ap` A list of arguments list of the `va_list` type that should be treated according to the format specification `fmt`

([I] : input, [O] : output)

RETURN VALUE

Non-negative value : Corresponding element's index (if the specified key string is found)
Negative value (error) : If the specified key string is not found

EXCEPTION

The member functions 2 and 3 throw an exception when it fails to allocate a local buffer.

2.4.5 key()

NAME

key() — Acquire the key string corresponding to the element number

SYNOPSIS

```
const char *key( size_t index ) const;
```

DESCRIPTION

`key()` gets the key string corresponding to the element's index specified by `index`. The element's index at the beginning of the array is zero.

If a value equal to or greater than the length of the array is given to `index`, it returns `NULL`.

PARAMETER

[I] `index` Array subscript of integer type `size_t`

([I] : input, [O] : output)

RETURN VALUE

An address of an internal buffer of a key string (normal end)
NULL (error, if the specified element's index equal to or greater than the length of an array)

2.4.6 keys()

NAME

keys() — A reference to the array object of the key string (read only)

SYNOPSIS

```
const tarray_tstring &keys() const;
```

DESCRIPTION

`keys()` returns a reference to an object of the key string array (tarray_tstring class: See the Basic Editon Manual) managed in the object. Member functions in the tarray_tstring class can be used by placing the "." operator immediately after this member function. The only available member functions in the tarray_tstring class are those that do not change the string, or in other words, tthose with the `const` attribute.

RETURN VALUE

A reference to an array object (tarray_tstring class) of a key string

---

2.4.7 values()

NAME

values() — A reference to the array object of the value (read only)

SYNOPSIS

```
const tarray<type> &values() const;
```

DESCRIPTION

`values()` returns a reference to an object of the array of the values (tarray template class: See §1) managed in the object. Member functions in the tarray template class can be used by placing the "." operator immediately after this member function. The only available member functions in the tarray template class are those that do not change the string, or in other words, tthose with the `const` attribute.

RETURN VALUE

A reference to an array object (tarray template class) of a value

---

2.4.8 swap()

NAME

swap() — Replace objects

SYNOPSIS

```
asarray &swap( asarray &sobj );
```

DESCRIPTION

`swap()` swaps the caller's content for that of object `sobj`.

PARAMETER

[I/O] `sobj` An instance of the class "asarray" that should swap its contents for this object

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

2.4.9 init()

NAME

init() — Initialize an object completely

SYNOPSIS

```
asarray &init();  ..... 1
asarray &init( const asarray &obj );  ..... 2
```

DESCRIPTION

`init()` initializes the caller's associative array.

The member function 1 initializes the caller completely. A memory area allocated to the associative array buffer or elements is released completely.

The member function 2 initializes the caller with the content of the specified object `obj` (copies the whole content of `obj` to the caller object).

PARAMETER

[I] `obj` An instance of the class "asarray" whose contents should be copied to this object

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

- Both functions throw an exception when they fail to allocate a local buffer.
- The member function 2 also throws an exception when it detects memory corruption.

2.4.10 assign()

NAME

assign() — Initialize an object and substitute an element

SYNOPSIS

```
asarray &assign( const char *key, const type &val );  ..... 1
asarray &assign( const asarray &src );  ..... 2
```

DESCRIPTION

The member function 1 initializes the caller's associative array with one specified element (a combination of the key and value).

The member function 2 assigns the content of the specified object `src` to the caller's associative array.

PARAMETER

[I] `key` A key string to be registered for the associative array

[I] `val` A reference to a scalar associated with the key string

[I] `src` An instance of the class "asarray" whose contents should be copied to this object

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

2.4.11 assign_keys()

NAME

assign_keys() — Assign multiple strings or arrays of strings to the key

SYNOPSIS

```
asarray &assign_keys( const char *key0, ... );  ..... 1
asarray &vassign_keys( const char *key0, va_list ap );  ..... 2
asarray &assign_keys( const char *const *keys );  ..... 3
asarray &assign_keys( const tarray_tstring &keys );  ..... 4
```

DESCRIPTION

`assign_keys()` sets the specified multiple strings `key0`, ... or the string array `keys` to the key of the caller's associative array. See the Basic Edition Manual for the tarray_tstring class.

The number of keys specified by the argument is set as the number of the associative arrays. When the elements in the associative array exceed the number of keys specified by the argument, they are deleted.

The variable argument in the member functions 1 and 2 and the pointer array `keys` in the member function 3 must be terminated with `NULL`.

PARAMETER

[I] `key0` A key string to be registered for the associative array
[I] `...` A variable number of arguments that contain key strings and are terminated by `NULL`
[I] `ap` A list of arguments of the `va_list` type that consists of key strings and is terminated by `NULL`
[I] `keys` An array of character strings whose last entry should be `NULL` for the member function 3

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

2.4.12 split_keys()

NAME

split_keys() — Split a string and assign to the key

SYNOPSIS

```
asarray &split_keys( const char *src_str, const char *delims,
                     bool zero_str, const char *quotations,
                     int escape, bool rm_escape );  ..... 1
asarray &split_keys( const char *src_str, const char *delims,
                     bool zero_str = false );  ..... 2
asarray &split_keys( const tstring &src_str, const char *delims,
                     bool zero_str, const char *quotations,
                     int escape, bool rm_escape );  ..... 3
asarray &split_keys( const tstring &src_str, const char *delims,
                     bool zero_str = false );  ..... 4
```

DESCRIPTION

`split_keys()` splits the string `src_str` by the delimiter character `delims` and assigns them to the key of the caller's associative array. For `delims`, it is possible to specify with such a simple character list as `" \t"` or `"[A-Z]"` or `"[^A-Z]"` used for regular expressions. Furthermore, a character class can be specified in `"[...]"`. See the description about the trim() member function of the tstring class in the Basic Edition Manual for specifiable character classes.

The number of keys obtained after splitting the string is set as the number of the associative arrays. (When the elements in the associative array exceed the number of keys obtained after splitting the string, they are deleted.)

Whether or not the string of zero length is allowed can be specified with `zero_str`. If `zero_str` is `false`, creating key element of the string of zero length is not allowed. If

`zero_str` is `true`, creating the key element of string of the zero length is allowed. (It can be used in the csv format.) If `zero_str` is not specified, `false` is assumed.

If you do not want to split a string enclosed by "specific characters" such as a quotation mark, specify "specific characters" in `quotations`. For example, to exclude strings enclosed by single quotation marks from splitting, specify as `"'"`.

Escape characters can be specified with `escape`. To remove escape characters after splitting, set `true` to `rm_escape`. However, escape characters enclosed by the characters specified by `quotations` are not removed.

If you can not get keys successfully with this member function alone, try to create a key string in the tarray_tstring class object and set the keys by using the assign_keys member function (§2.4.11).

PARAMETER

[I]	`src_str`	A character string to be split
[I]	`delims`	A character string that consists of delimiters
[I]	`zero_str`	True if you permit the function to produce a zero-sized key string (i.e., a key string containing no character).
[I]	`quotations`	A character string that consists of quotation marks
[I]	`escape`	An escape character
[I]	`rm_escape`	True if the escape character should be removed

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

2.4.13 append()

NAME

append() — Append an element

SYNOPSIS

```
asarray &append( const char *key, const type &val );  ..... 1
asarray &append( const asarray &src );  ..... 2
```

DESCRIPTION

The member function 1 appends one specified element (a combination of the key and value) to the caller's associative array. The member function 2 assigns the content of the specified object `src` to the caller's associative array.

If the specified key overlaps with the existing key, a warning is output to stderr upon execution and no processing is performed.

PARAMETER

[I]	`key`	A key string to be appended to the associative array
[I]	`val`	A reference to a scalar being associated with the key string
[I]	`src`	An instance of the class "asarray" whose contents should be copied to this object

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

2.4.14 insert()

NAME

insert() — Insert an element

SYNOPSIS

```
asarray &insert( const char *key,
                 const char *newkey, const type &newval );  ..... 1
asarray &insert( const char *key, const asarray &src );  ..... 2
```

DESCRIPTION

The member function 1 inserts one specified element (a combination of the key and value) followed by the position of the element corresponding the key `key` in the caller's associative array.

The member function 2 inserts the content of the specified object `src` followd by the position of the element corresponding the key `key` in the caller's associative array.

If the specified key overlaps with the existing key, a warning is output upon execution and no processing is performed.

PARAMETER

[I]	`key`	A key string specifying an element of this array before which a new element should be inserted
[I]	`newkey`	A key string to be inserted to the associative array
[I]	`newval`	A reference to a scalar being associated with the key string
[I]	`src`	An instance of the class "asarray" whose contents should be copied to this object

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

2.4.15 erase()

NAME

erase() — Erase an element (a set of a key and a value)

SYNOPSIS

```
asarray &erase();  ..... 1
asarray &erase( const char *key, size_t num_elements = 1 );  ..... 2
```

DESCRIPTION

`erase()` erases elements in the caller's associative array.

The member function 1 erases all elements in the caller's associative array. (The length of the array becomes 0.)

The member function 2 erases `num_elements` number of elements starting from the element corresponding to the key specified by `key`. If `num_elements` is not specified, it erases one element.

The array become shorter as more elements are erased.

PARAMETER

[I]	`key`	A key string specifying the first element to be removed
[I]	`num_elements`	The number of elements to be removed

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

2.4.16 clean()

NAME

clean() — Padding of the total existing values in an associated array by arbitrary ones

SYNOPSIS

```
asarray &clean(const type &one);
```

DESCRIPTION

`clean()` performs the padding of all elements in the caller's associative array with the value `one`. The length of the array does not change even after `clean()` is executed.

PARAMETER

[I]	`one`	A reference to a scalar that should be written to all entries of the associative array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer for elements.

2.4.17 rename_a_key()

NAME

rename_a_key() — Change a key string

SYNOPSIS

```
asarray &rename_a_key( const char *org_key, const char *new_key );
```

DESCRIPTION

`rename_a_key()` changes the key string `org_key` in the caller's associative array to the string specified by `new_key`.

If the key string specified by `org_key` is not found in the associative array, or if `new_key` overlaps with the existing key, an error message is output to the standard error output.

PARAMETER

[I] `org_key` Original key string replaced by `new_key`

[I] `new_key` A new key string replacing `org_key`

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

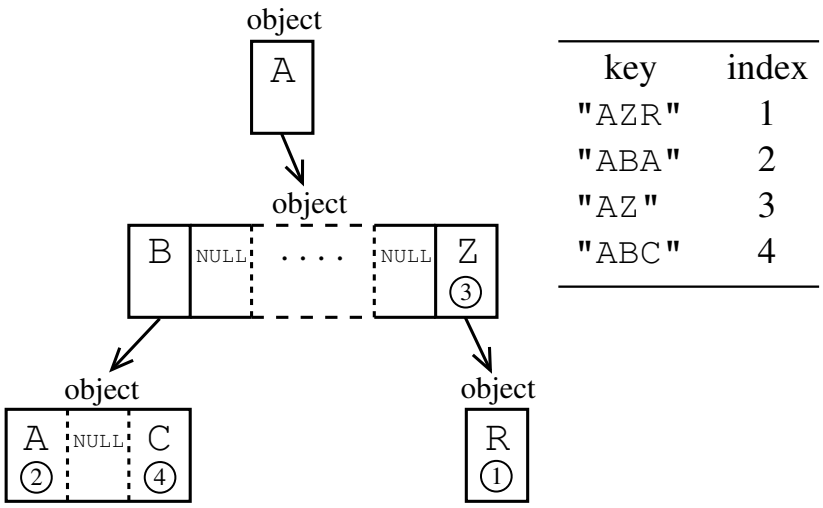
EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

---

3 CTINDEX Class

The ctindex class manages the relationship between the key string and the index. Once the relationship between a string (key) and a number (index) is registered, the index can be retrieved quickly by using key string. Its behavior is similar to that of hash, but no collision occurs because this class manages the relationship between the key string and the index with a dictionary.



key	index
"AZR"	1
"ABA"	2
"AZ"	3
"ABC"	4

Figure 1: Outline of ctindex Class. Each node is a ctindex-class object and a part of the tree structure. The circled number is the index stored. The right table shows the set of key strings and indexes.

A working principle of the ctindex class is shown in Figure 1. As shown in the figure, when the key string is registered, multiple ctindex objects constitute a tree structure and the index is saved in the buffer of each node (object). Therefore, the time it takes to retrieve the index with the key string is proportional to the length of the string without exception.

To use the ctindex class, write the following code followed by the user code:

```
#include <sli/ctindex.h>
```

Additionally, write "`using namespace sli;`" in the code if the namespace declaration is required.

3.1 How to Create an Object

To create an object of the ctindex class, no arguments are required. Create it normally as shown below:

```
#include <sli/ctindex.h>
using namespace sli;

int main()
{
    ctindex my_index;
```

3.2 List of Member Functions

Member functions are listed in Table 3.

	Function Name	Description
§3.3.1	`=`	Substitute the ctindex object
§3.4.1	`init()`	Initialize of an object completely
§3.4.2	`append()`	Append a pair of a key and an index
§3.4.3	`update()`	Update an index
§3.4.4	`erase()`	Erase a pair of a key and an index
§3.4.5	`index()`	Acquire an index value

Table 3: List of Member Functions Available in ctindex Class

3.3 Operators

3.3.1 =

NAME

= — Substitute the ctindex object

SYNOPSIS

```
ctindex &operator=(const ctindex &obj);
```

DESCRIPTION

This operator substitutes the ctindex-class object specified by the right side of the operator (argument) into the object itself.

PARAMETER

[I] `obj` An object belonging to the `ctindex` class

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code substitutes the ctindex object `my_idx` into the object `my_idx1` and prints each index of the key string `rose` and `cosmos` to stdout. For more information about `index()` and `append()`, see §3.4.5 and §3.4.2.

```
stdstreamio sio;

ctindex my_idx;
ctindex my_idx1;

my_idx.append("rose",0);
my_idx.append("cosmos",1);

my_idx1 = my_idx;
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("rose"));
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("cosmos"));
```

Output:

```
my_idx1.index...  [0]
my_idx1.index...  [1]
```

3.4 Member Functions

General Notice

Multiple indexes can be assigned to one key string. The search of the key string becomes slow in proportion to the number of indexes.

When the key string is not found when retrieving the index, `-1` is returned.

The index can be retrieved from the key string, but the key string cannot be retrieved from the index.

3.4.1 init()

NAME

init() — Initialize of an object completely

SYNOPSIS

DESCRIPTION

`init()` initializes an object.

The member function 1 initializes an object completely, and a memory area allocated to a dictionary buffer in the object is released completely.

The member function 2 initializes with the content of the ctindex class object `obj`.

PARAMETER

[I] `obj` An object belonging to the class "ctindex"
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

- Both functions throw an exception when they fail to allocate a local buffer.
- The member function 2 also throws an exception when it detects memory corruption.

EXAMPLE

The following code initializes the ctindex-class object `my_idx1` by `my_idx` and prints the index to stdout. For more information about `index()` and `append()`, see §3.4.5 and §3.4.2.

```
stdstreamio sio;

ctindex my_idx;
ctindex my_idx1;

my_idx.append("Morning glory",0);
my_idx.append("Gentiana",1);

my_idx1.init(my_idx);
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("Morning glory"));
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("Gentiana"));
```

Output:

```
my_idx1.index...  [0]
my_idx1.index...  [1]
```

3.4.2 append()

NAME

append() — Append a pair of a key and an index

SYNOPSIS

```
int append( const char *key, size_t index );
```

DESCRIPTION

`append()` appends the key string `key` and the corresponding index `index` to a ctindex object.

PARAMETER

[I]	`key`	A key string to be appended
[I]	`index`	Value of the index specifying the key string

([I] : input, [O] : output)

RETURN VALUE

0	:	Normal end
Negative value(error)	:	If the same combination of a key and an index exist If a key is `NULL`

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code appends `"Pansy"`, `2` and `"Violet"`, `4` to the ctindex-class object `my_idx` and prints the indexes of `"Pansy"` and `"Violet"` to stdout for confirmation. For the information about `index()`, see §3.4.5:

```
stdstreamio sio;

ctindex my_idx;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Violet"));
```

Output:

```
my_idx.index...  [2]
my_idx.index...  [4]
```

3.4.3 update()

NAME

update() — Update an index

SYNOPSIS

```
int update( const char *key, size_t current_index, size_t new_index );
```

DESCRIPTION

`update()` changes a value of the index corresponding to the key string `key`.

PARAMETER

[I]	`key`	A key string
[I]	`current_index`	Current value of the index for the key
[I]	`new_index`	Updated value of the index for the key

([I] : input, [O] : output)

RETURN VALUE

0	:	Normal end
Negative value(error)	:	If a key is `NULL`

EXAMPLE

The following code changes the index of the key `"Pansy"` in the ctindex-class object `my_idx` from `2` to `1` and prints the index to stdout twice before and after `update()` for confirmation. For more information about `index()`, see §3.4.5.

```
stdstreamio sio;

ctindex my_idx;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));

my_idx.update("Pansy", 2, 1);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));
```

Output:

```
my_idx.index...  [2]
my_idx.index...  [1]
```

3.4.4 erase()

NAME

erase() — Erase a pair of a key and an index

SYNOPSIS

```
int erase( const char *key, size_t index );
```

DESCRIPTION

`erase()` erases the key string `key` and the corresponding index `index`.

PARAMETER

[I]	`key`	A key string to be removed
[I]	`index`	Value of the index for the key

([I] : input, [O] : output)

RETURN VALUE

Value more than 0	:	Normal end
Negative value (error)	:	If the combination of the specified key and index does not exist If the specified key does not exist If the specified key is `NULL`

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code erases the set of the key `"Pansy"` and the value `2` in the ctindex-class object `my_idx` and prints the index to stdout after `erase()` for confirmation. For more information about `index()` and `append()`, see §3.4.5 and §3.4.2.

```
stdstreamio sio;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
my_idx.append("Pansy",6);

my_idx.erase("Pansy", 2);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));
```

Output:
`my_idx.index...  [6]`

3.4.5 index()

NAME
index() — Acquire an index value

SYNOPSIS
```
ssize_t index( const char *key, int index_of_index = 0 ) const;
```

DESCRIPTION
`index()` gets on index value of the key string `key`.

If the key string `key` has multiple indexes, set n to `index_of_index` to get n-th value of the index. `index_of_index` starts from zero.

PARAMETER

[I]	`key`	A key string
[I]	`index_of_index`	Subscript of integer type specifying one of the several indices for the key

([I] : input, [O] : output)

RETURN VALUE

Non-negative value	:	A value of index
Negative value(error)	:	If the index corresponding to the specified key does not exist

EXAMPLE
The following code registers the same key with a different index to the ctindex-class object `my_idx` and prints the index to stdout. For more information about `append()`, see §3.4.2.

```
stdstreamio sio;

ctindex my_idx;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
my_idx.append("Pansy",6);

int i = 0 ;
while ( 0 < my_idx.index("Pansy",i) ) {
    sio.printf("my_idx.index[Pansy]... [%zd]\n", my_idx.index("Pansy",i));
    i++;
}
```

Output:

```
my_idx.index[Pansy]...  [2]
my_idx.index[Pansy]...  [6]
```

4 MDARRAY Class

The mdarry class can handle multidimensional arrays of types in C or structures.

It has features as follows:

- It has two operating modes:"Automatic Resize Mode", in which the memory area is allocated automatically as necessary when users access an element of arrays and "Manual Resize Mode", which is suitable for handling an image data.
- Operators "`+`", "`-`", "`*`", "`/`", "`+=`", "`-=`", "`*=`", "`/=`", and "`=`" are available for scalar values and the whole array.
- Mathematic functions (`sin()`, `log()`, etc.) to carry out an operation to the whole array are prepared. (Almost all functions defined in math.h of libc are available.)

To use the classes, write the following code followed by the user code:

```
#include <sli/mdarray.h>
#include <sli/mdarray_math.h>
```

`mdarray_math.h` is not necessary when only using classes, but it is necessary when using mathematic functions on the whole array. Additionally, write `using namespace sli;` in the code when namespace declaration is required.

Brief example of use is shown following.

```
#include <sli/stdstreamio.h>
#include <sli/mdarray.h>
using namespace sli;

int main()
{
    stdstreamio sio;
    mdarray my_mdarr(INT_ZT);   /* declare an array of integer */

    my_mdarr.i(0) = 10;         /* the array size is automatically set */
    my_mdarr.i(1) = 20;         /* access an integer
                                    via the member function i() */

    my_mdarr.i(0,1) = 30;       /* extended to the 2x2(2-dimension) array */

    return 0;
}
```

4.1 How to Create an Object

Create a mdarray class object by using the following constructor:

Automatic Resize Mode

Manual Resize Mode

```
mdarray( ssize_t sz_type, size_t naxis0, size_t naxis1, size_t naxis2 );    5
mdarray( ssize_t sz_type, const size_t naxisx[], size_t ndim );  ..... 6
```

The member functions 3-5 are used to create one-, two- and three- dimensional arrays, respectively. The member function 6 is used to create n dimensional array. As for the member functions 1 and 2, while the number of dimensions and the length of the array are both 0 immediately after the object is created, the number of dimensions and the length of the array are extended automatically when the element is accessed.

Give the integer (type-specfic) that represents type in C to the argument `sz_type`. The constant for this integer is defined in "sli/size_types.h" for your use (Table 4). Use the constant name instead of using the actual value directly in the code.

Data Type in C	Constant Identifier	Value	Description
float	`FLOAT_ZT`	`-4`	Single-precision floating-point number
double	`DOUBLE_ZT`	`-8`	Souble-precision floating-point number
fcomplex	`FCOMPLEX_ZT`	`-7`	Single-precision floating-point complex Number (**unusable, unaccomplished**)
dcomplex	`DCOMPLEX_ZT`	`-15`	Double-precision floating-point complex Number (**unusable, unaccomplished**)
unsigned char	`UCHAR_ZT`	`1`	Unsigned 1-byte integer
short	`SHORT_ZT`	system-dependent	Signed integer
int	`INT_ZT`	system-dependent	Signed integer
long	`LONG_ZT`	system-dependent	Signed integer
long long	`LLONG_ZT`	system-dependent	Signed integer
int16_t	`INT16_ZT`	`2`	Signed 2-byte integer
int32_t	`INT32_ZT`	`4`	Signed 4-byte integer
int64_t	`INT64_ZT`	`8`	Signed 8-byte integer
size_t	`SIZE_ZT`	system-dependent	Snsigned integer
ssize_t	`SSIZE_ZT`	system-dependent	Signed integer
bool	`BOOL_ZT`	system-dependent	Boolean
uintptr_t	`UINTPTR_ZT`	system-dependent	Unsigned integer corresponding To the address width

Table 4: List of Constants Defined in sli/size_types.h

To handle an array of the structure, give the byte size of the structure to `sz_type`. If the argument `sz_type` is not given (in case of the member function 1), it is assumed that `UCHAR_ZT` is given.

The mdarray class has two kinds of operating modes: (1)Automatic Resize Mode and (2)Manual Resize Mode. In the Automatic Resize Mode, the number of dimensions and size of arrays are extended automatically as necessary when you access the element of the array by using `assign()`, etc., or carry out an operation of the array by using the operators "`+=`", "`-=`", etc. On the other hand, in the Manual Resize Mode, the buffer size is not changed unless expressly specified to be resized (thus, this mode is suitable for handling an image data).

The operating mode is decided when the object is initialized by the constructors or the init() member function (§4.5.24). It works in the Automatic Resize Mode when the number of the arrays is not given and in the Manual Resize Mode when the number of the arrays is given upon initialization.

4.2 Mathematic Functions

The available mathematic functions are shown in Table 5.

4.3 List of Member Functions

A list of member functions is shown in Table 6.

Function Prototype	Description
`mdarray cbrt( const mdarray &obj );`	Cubic root
`mdarray sqrt( const mdarray &obj );`	Square root
`mdarray asin( const mdarray &obj );`	Arc sine
`mdarray acos( const mdarray &obj );`	Arc cosine
`mdarray atan( const mdarray &obj );`	Arc tangent
`mdarray acosh( const mdarray &obj );`	Arc hyperbolic cosine
`mdarray asinh( const mdarray &obj );`	Arc hyperbolic sine
`mdarray atanh( const mdarray &obj );`	Arc hyperbolic tangent
`mdarray exp( const mdarray &obj );`	Exponential function with base e
`mdarray exp2( const mdarray &obj );`	Exponential function with base 2
`mdarray expm1( const mdarray &obj );`	Exponent of argument minus 1
`mdarray log( const mdarray &obj );`	Natural logarithmic function
`mdarray log1p( const mdarray &obj );`	Logarithm of 1 plus argument
`mdarray log10( const mdarray &obj );`	Logarithm with base 10
`mdarray sin( const mdarray &obj );`	Sine
`mdarray cos( const mdarray &obj );`	Cosine
`mdarray tan( const mdarray &obj );`	Tangent
`mdarray sinh( const mdarray &obj );`	Hyperbolic sine
`mdarray cosh( const mdarray &obj );`	Hyperbolic cosine
`mdarray tanh( const mdarray &obj );`	Hyperbolic tangent
`mdarray erf( const mdarray &obj );`	Error function
`mdarray erfc( const mdarray &obj );`	Complementary error function
`mdarray ceil( const mdarray &obj );`	Smallest integer not less than argument
`mdarray floor( const mdarray &obj );`	Largest integer not greater than argument
`mdarray round( const mdarray &obj );`	Round a number to the nearest integer
`mdarray trunc( const mdarray &obj );`	Truncate a number to the next nearest integer towards 0
`mdarray fabs( const mdarray &obj );`	Absolute value

`mdarray hypot( const mdarray &obj, float v );`
`mdarray hypot( const mdarray &obj, double v );`
`mdarray hypot( float v, const mdarray &obj );`
`mdarray hypot( double v, const mdarray &obj );`
`mdarray hypot( const mdarray &src0,`
`const mdarray &src1 );`
	Euclidean distance function

`mdarray pow( const mdarray &obj, float v );`
`mdarray pow( const mdarray &obj, double v );`
`mdarray pow( float v, const mdarray &obj );`
`mdarray pow( double v, const mdarray &obj );`
`mdarray pow( const mdarray &src0,`
`const mdarray &src1 );`
	Power

`mdarray fmod( const mdarray &obj, float v );`
`mdarray fmod( const mdarray &obj, double v );`
`mdarray fmod( float v, const mdarray &obj );`
`mdarray fmod( double v, const mdarray &obj );`
`mdarray fmod( const mdarray &src0,`
`const mdarray &src1 );`
	Modulo arithmetic

`mdarray remainder( const mdarray &obj, float v );`
`mdarray remainder( const mdarray &obj, double v );`
`mdarray remainder( float v, const mdarray &obj );`
`mdarray remainder( double v, const mdarray &obj );`
`mdarray remainder( const mdarray &src0,`
`const mdarray &src1 );`
	Remainder

Table 5: List of Available Mathematic Functions

	Function Name	Description	Operating Mode Support
§4.4.1	`=`	Substitute an array (initialization of an object)	
§4.4.2	`=`	Substitute a scalar value	
§4.4.3	`+=`	Add an array to itself	◯
§4.4.4	`+=`	Add a scalar value to itself	
§4.4.5	`-=`	Subtract an array from itself	◯
§4.4.6	`-=`	Subtract a scalar value from itself	
§4.4.7	`*=`	Multiply itself by an array	◯
§4.4.8	`*=`	Multiply itself by a scalar value	
§4.4.9	`/=`	Divide itself by an array	◯
§4.4.10	`/=`	Divide itself by a scalar value	
§4.4.11	`+`	Return the object which stores the result of adding an array to itself	
§4.4.12	`+`	Return the object which stores the result of adding a scalar value to itself	
§4.4.13	`-`	Return the object which stores the result	
§4.4.14		of subtracting an array from itself	
	`-`	Return the object which stores the result of subtracting a scalar value from itself	
§4.4.15	`*`	Return the object which stores the result of multiplying itself by an array	
§4.4.16	`*`	Return the object which stores the result of multiplying itself by a scalar value	
§4.4.17	`/`	Return the object which stores the result of dividing itself by an array	
§4.4.18	`/`	Return the object which stores the result of dividing itself by a scalar value	
§4.4.19	`==`	Compare	
§4.4.20	`!=`	Compare (negative form)	
§4.5.1	`size_type()`	Integer representing a data type	
§4.5.2	`bytes()`	Number of bytes of an element	
§4.5.3	`dim_length()`	Number of dimensions of an array	
§4.5.4	`length()`	Number of elements (in a dimension)	
§4.5.5	`byte_length()`	Total byte size of elements (in a dimension) in an array	
§4.5.6	`col_length()`	The length of the array's column	
§4.5.7	`row_length()`	The length of the array's row	
§4.5.8	`layer_length()`	The layer number of the array	
§4.5.9	`f(), ...`	A reference to specified value of the element	◯
§4.5.10	`f_cs(), ...`	Read specified value of the element	
§4.5.11	`dvalue()`	The value of the element converted into double type	
§4.5.12	`lvalue(),`	The value of the element converted into long	
	`llvalue()`	or long long type	

Table 6: List of Member Functions Available in mdarray Class (cont'd)

	Function Name	Description	Operating Mode Support
§4.5.13	`default_value_ptr()` `assign_default()`	The acquisition and the setting of the initial value	
§4.5.14	`auto_resize()`, `set_auto_resize()`	The acquisition and the setting of the resize mode	
§4.5.15	`rounding()` `set_rounding()`	The acquisition and the setting of the rounding off possibility	
§4.5.16	`dprint()`	Output of the object information to the stderr output(for user's debug)	
§4.5.17	`data_ptr()`	The acquisition of the specified element's address	
§4.5.18	`register_extptr()`	Register a user pointer variable	
§4.5.19	`get_elements()`	Copy the array itself to the user's buffer	
§4.5.20	`put_elements()`	Copy the array in the user's buffer to the array itself	
§4.5.21	`getdata()`	Copy the array itself to the user's buffer	
§4.5.22	`putdata()`	Copy the array in the user's buffer to the array itself	
§4.5.23	`reverse_endian()`	Reverse the endian if necessary	
§4.5.24	`init()`	The initialization of the array	
§4.5.25	`assign()`	Substitute a value for an element(high level)	◯
§4.5.26	`put()`	Set a value to an arbitrary element's point	◯
§4.5.27	`swap()`	Replace values between elements	
§4.5.28	`move()`	Copy values between elements	
§4.5.29	`cpy()`	Copy values between elements (with automatic expansion)	
§4.5.30	`insert()`	Insert an element	
§4.5.31	`crop()`	Extract an element	
§4.5.32	`erase()`	Erase an element	
§4.5.33	`resize()`	Change the length of the array	
§4.5.34	`resizeby()`	Change the length of the array relatively	
§4.5.35	`increase_dim()`	The expansion of the dimension number	
§4.5.36	`decrease_dim()`	The reduction of the dimension number	
§4.5.37	`swap()`	Replace the object by another one	
§4.5.38	`convert()`	Convert the type of the array	
§4.5.39	`ceil()`	Raise decimals to the next whole number in a double type value	
§4.5.40	`floor()`	Devalue decimals in a double type value	
§4.5.41	`round()`	Round off decimals in a double type value	
§4.5.42	`trunc()`	Omit decimals in a double type value	
§4.5.43	`abs()`	Absolute value of all elements	
§4.5.44	`compare()`	Compare array objects	

Table 6: List of Member Functions Available in mdarray Class (cont'd)

	Function Name	Description	Operating Mode Support
§4.5.45	`copy()`	Copy an array into another object	
§4.5.46	`copy()`	Copy a part of an array into another object (for image data)	
§4.5.47	`cut()`	Cut all values in an array and copy them into another object	
§4.5.48	`cut()`	Cut a part of values in an array and copy them into another object (for image data)	
§4.5.49	`clean()`	Padding existing values in an array by default ones (for image data)	
§4.5.50	`fill()`	Rewrite element values (for image data)	
§4.5.51	`add()`	Add element values (for image data)	
§4.5.52	`multiply()`	Multiply element values (for image data)	
§4.5.53	`paste()`	Paste up an array object (for image data)	
§4.5.54	`add()`	Add an array object (for image data)	
§4.5.55	`subtract()`	Subtract an array object (for image data)	
§4.5.56	`multiply()`	Multiply an array object (for image data)	
§4.5.57	`divide()`	Divide an array object (for image data)	

Table 6: List of Member Functions Available in mdarray Class

4.4 Operators

4.4.1 =

NAME

= — Initialize of an object

SYNOPSIS

```
mdarray &operator=(const mdarray &obj);
```

DESCRIPTION

This operator initializes the object itself and copies all the contents and attributes (the length of arrays and the data types, etc.) of `obj` to the object itself.

PARAMETER

[I] `obj` An object belonging to the class "mdarray"
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code initializes the object `my_mdarr` of the mdarray class by `area_mdarr` and prints the result to stdout. For more information about `ll()` and `length()`, see §4.5.9 and §4.5.4.

```
stdstreamio sio;

mdarray my_mdarr;
mdarray area_mdarr(LLONG_ZT);
area_mdarr.ll(0) = 17090000;
area_mdarr.ll(1) = 9980000;
area_mdarr.ll(2) = 9620000;

my_mdarr = area_mdarr;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%lld]\n", i, my_mdarr.ll(i));
}
```

Output:

```
my_mdarr value[0]...  [17090000]
my_mdarr value[1]...  [9980000]
my_mdarr value[2]...  [9620000]
```

4.4.2 =

NAME

= — Substitute of a scalar value

SYNOPSIS

```
mdarray &operator=(double v);  ..... 1
mdarray &operator=(long long v);  ..... 2
mdarray &operator=(long v);  ..... 3
mdarray &operator=(int v);  ..... 4
```

DESCRIPTION

This operator substitutes the value (scalar value) specified by the right side of the operator (argument) into the object itself. The array size is not extended automatically, so it is necessary to set the number of elements and to reserve the buffer area in advance.

PARAMETER

[I] `v` A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code substitutes the scalar value, `125`, into the object `mdarr` with a one-dimension array and prints the result to stdout. For more information about `length()` and `c()`, see §4.5.4 and §4.5.9.

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 2);

my_mdarr = 125;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%hhu]\n", i, my_mdarr.c(i));
}
```

Output:

```
my_mdarr value[0]...  [125]
my_mdarr value[1]...  [125]
```

4.4.3 +=

NAME

+= — Add an array to itself

SYNOPSIS

```
mdarray &operator+=(const mdarray &obj);
```

DESCRIPTION

This operator adds the object array of the mdarray class specified by the right side of the operator (argument) to the object itself. When object with the different data type from that of the object itself is specified, cast operations are executed just like a normal scalar operation.

In the Automatic Resize Mode, the array size is extended automatically if each dimension size of `obj` is larger than that of the object itself.

PARAMETER

[I] `obj` An object belonging to the class "mdarray"
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

4.4.4 +=

NAME

+= — Add a scalar value to itself

SYNOPSIS

```
mdarray &operator+=(double v);
mdarray &operator+=(long long v);
mdarray &operator+=(long v);
mdarray &operator+=(int v);
```

DESCRIPTION

This operator adds the scalar value specified by the right side of the operator (argument) to all elements of the object itself. When the data type of the argument is different from that of the object itself, cast operations are executed just like a normal scalar operation.

PARAMETER

[I] `v` A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code adds the scalar value, `50`, to the object `my_mdarr` of the mdarray class and prints the result to stdout. For more information about `length()` and `c()`, see §4.5.4 and §4.5.9.

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 2);
my_mdarr = 25;
my_mdarr += 50;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%hhu]\n", i, my_mdarr.c(i));
}
```

Output:

```
my_mdarr value[0]...  [75]
my_mdarr value[1]...  [75]
```

4.4.5 -=

NAME

−= — Subtract an array from itself

SYNOPSIS

```
mdarray &operator-=(const mdarray &obj);
```

DESCRIPTION

This operator subtracts the object array of the mdarray class specified by the right side of the operator (argument) from the object itself. When the object, with the different data type from that of the object itself is specified, cast operations are executed just like a normal scalar operation.

In the Automatic Resize Mode, the array size is extended automatically if each dimension size of `obj` is larger than that of the object itself.

PARAMETER

[I] `obj` An object belonging to the class "mdarray"
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code subtracts the object `subst_mdarr` from the object `my_mdarr` of the mdarray class and prints the result to stdout. For more information about `length()` and `c()`, see §4.5.4 and §4.5.9.

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 2);
my_mdarr = 100;

mdarray subst_mdarr(UCHAR_ZT, 2);
subst_mdarr.c(0) = 10;
subst_mdarr.c(1) = 20;

my_mdarr -= subst_mdarr;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%hhu]\n", i, my_mdarr.c(i));
}
```

Output:

```
my_mdarr value[0]...  [90]
my_mdarr value[1]...  [80]
```

4.4.6 -=

NAME

−= — Subtract a scalar value from itself

SYNOPSIS

```
mdarray &operator-=(double v);
mdarray &operator-=(long long v);
mdarray &operator-=(long v);
mdarray &operator-=(int v);
```

DESCRIPTION

This operator subtracts the scalar value specified by the right side of the operator(argument) from all elements of the object itself. In case that the data type of the argument is different from that of the object itself, cast operations are executed like a normal scalar operation.

PARAMETER

[I] `v` a real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

4.4.7 *=

NAME

∗= — Multiply itself by an array

SYNOPSIS

```
mdarray &operator*=(const mdarray &obj);
```

DESCRIPTION

This operator multiplies the object itself by the object array of the mdarray class specified by the right side of the operator (argument). When the object with the different data type from that of the object itself is specified, cast operations are executed just like a normal scalar operation.

In the Automatic Resize Mode, the array size is extended automatically if each dimension size of `obj` is larger than that of the object itself.

PARAMETER

[I] `obj` An object belonging to the class "mdarray"
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code multiplies the object `my_mdarr` of the mdarray class by the object `mdarrPlus` and prints the result to stdout. For more information about `length()` and `l()`, see §4.5.4 and §4.5.9.

```
stdstreamio sio;

mdarray my_mdarr(LONG_ZT, 2);
my_mdarr = 50;

mdarray multi_mdarr(LONG_ZT);
multi_mdarr.l(0) = 10;
multi_mdarr.l(1) = 20;
multi_mdarr.l(2) = 30;

my_mdarr *= multi_mdarr;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%ld]\n", i, my_mdarr.l(i));
}
```

Output:

```
my_mdarr value[0]...  [500]
my_mdarr value[1]...  [1000]
```

4.4.8 *=

NAME

*= — Multiply itself by a scalar value

SYNOPSIS

```
mdarray &operator*=(double v);
mdarray &operator*=(long long v);
mdarray &operator*=(long v);
mdarray &operator*=(int v);
```

DESCRIPTION

This operator multiplies all elements of the object itself by the scalar value specified by the right side of the operator (argument). When the data type of the argument is different from that of the object itself, cast operations are executed just like a normal scalar operation.

PARAMETER

[I] v A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

4.4.9 /=

NAME

/= — Divide itself by an array

SYNOPSIS

```
mdarray &operator/=(const mdarray &obj);
```

DESCRIPTION

This operator divides the object itself by the object array of the mdarray class specified by the right side of the operator (argument). When the object with the different data type from that of the object itself is specified, cast operations are executed just like a normal scalar operation.

In cthe Automatic Resize Mode, the array size is extended automatically if each dimension size of `obj` is larger than that of the object itself.

PARAMETER

[I] `obj` An object belonging to the class "mdarray"
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code divides the object `my_mdarr` of the mdarray class by the object `div_mdarr` and prints the result to stdout. For more information about `length()` and `l()`, see §4.5.4 and §4.5.9.

```
stdstreamio sio;

mdarray my_mdarr(LONG_ZT, 2);
my_mdarr = 50;

mdarray div_mdarr(LONG_ZT);
div_mdarr.l(0) = 1;
div_mdarr.l(1) = 2;
div_mdarr.l(2) = 5;

my_mdarr /= div_mdarr;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%ld]\n", i, my_mdarr.l(i));
}
```

Output:

```
my_mdarr value[0]...  [50]
my_mdarr value[1]...  [25]
```

4.4.10 /=

NAME

/= — Divide itself by a scalar value

SYNOPSIS

```
mdarray &operator/=(double v);
mdarray &operator/=(long long v);
mdarray &operator/=(long v);
mdarray &operator/=(int v);
```

DESCRIPTION

This operator divides all elements of the object itself by the scalar value specified by the right side of the operator (argument). When the data type of the argument is different from that of the object itself, cast operations are executed just like a normal scalar operation.

PARAMETER

[I] `v` A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

4.4.11 +

NAME

+ — Return the object that stores the result of adding an array to itself

SYNOPSIS

```
mdarray operator+(const mdarray &obj);
```

DESCRIPTION

This operator returns the object that stores the result of adding the object array of the mdarray class specified by the right side of the operator (argument) to the object itself. The argument is the base class (mdarray class). Thus, the object with the different data type from that of the object itself can be specified to the argument and cast operations are executed just like a normal scalar operation.

The returned operation mode and rounding attribute are identical to those of the object itself.

PARAMETER

[I] `obj` A object belonging to the class "mdarray"
([I] : input, [O] : output)

RETURN VALUE

An object including the calculation result

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

4.4.12 +

NAME

+ — Return the object that stores the result of adding a scalar value to itself

SYNOPSIS

```
mdarray operator+(double v);
mdarray operator+(float v);
mdarray operator+(long long v);
mdarray operator+(long v);
mdarray operator+(int v);
```

DESCRIPTION

This operator returns the object that stores the result of adding the scalar value specified by the right side of the operator (argument) to all elements of the object itself. When the data type of the argument is different from that of the object itself, cast operations are executed just like a normal scalar operation.

The returned operation mode and rounding attributes are identical to those of the object itself.

PARAMETER

[I] `v` A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

4.4.13 -

NAME

− — Return the object that stores the result of subtracting an array from itself

SYNOPSIS

```
mdarray operator-(const mdarray &obj);
```

DESCRIPTION

This operator returns the object that stores the result of subtracting the object array of the mdarray class specified by the right side of the operator (argument) from the object itself. The argument is the base class (mdarray class). Thus, with the different data type from that of the object itself can be specified to the argument and cast operations are executed just like a normal scalar operation.

The returned operation mode and rounding attribute are identical to those of the object itself.

PARAMETER

[I] `obj` A object belonging to the class “mdarray”
([I] : input, [O] : output)

RETURN VALUE

An object including the calculation result

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

4.4.14 -

NAME

− — Return the object that stores the result of subtracting a scalar value from itself

SYNOPSIS

```
mdarray operator-(double v);
mdarray operator-(float v);
mdarray operator-(long long v);
mdarray operator-(long v);
mdarray operator-(int v);
```

DESCRIPTION

This operator returns the object that stores the result of subtracting the scalar value specified by the right side of the operator (argument) from all elements of the object itself. When the data type of the argument is different from that of the object itself, cast operations are executed just like a normal scalar operation.

The returned operation mode and rounding attributes are identical to those of the object itself.

PARAMETER

[I] `v` A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

4.4.15 *

NAME

∗ — Return the object that stores the result of multiplying itself by an array

SYNOPSIS

```
mdarray operator*(const mdarray &obj);
```

DESCRIPTION

This operator returns the object that stores the result of multiplying the object itself by the object array of the mdarray class specified by the right side of the operator (argument). The argument is the base class (mdarray class). Thus, the object with the different data type from that of the object itself can be specified to the argument and cast operations are executed just like a normal scalar operation.

The returned operation mode and rounding attribute are identical to those of the object itself.

PARAMETER

[I] `obj` A object belongings to the class “mdarray”
([I] : input, [O] : output)

RETURN VALUE

An object including the calculation result

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

4.4.16 *

NAME

∗ — Return the object that stores the result of multiplying itself by a scalar value

SYNOPSIS

```
mdarray operator*(double v);
mdarray operator*(float v);
mdarray operator*(long long v);
mdarray operator*(long v);
mdarray operator*(int v);
```

DESCRIPTION

This operator returns the object that stores the result of multiplying all elements of the object itself by the scalar value specified by the right side of the operator (argument). When the data type of the argument is different from that of the object itself, cast operations are executed like a normal scalar operation.

The returned operation mode and rounding attribute are identical to those of the object itself.

PARAMETER

[I] `v` A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

4.4.17 /

NAME

/ — Return the object that stores the result of dividing itself by an array

SYNOPSIS

```
mdarray operator/(const mdarray &obj);
```

DESCRIPTION

This operator returns the object that stores the result of dividing the object itself by the object array of the mdarray class specified by the right side of the operator (argument). The argument is the base class (mdarray class). Thus, the object with the different data type from that of the object itself can be specified to the argument and cast operations are executed just like a normal scalar operation.

The returned operation mode and rounding attributes are identical to those of the object itself.

PARAMETER

[I] `obj` An object that belonging to the class “mdarray”
([I] : input, [O] : output)

RETURN VALUE

An object including the calculation result

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

4.4.18 /

NAME

/ — Return the object that stores the result of dividing itself by a scalar value

SYNOPSIS

```
mdarray operator/(double v);
mdarray operator/(float v);
mdarray operator/(long long v);
mdarray operator/(long v);
mdarray operator/(int v);
```

DESCRIPTION

This operator returns the object that stores the result of dividing all elements of the object itself by the scalar value specified by the right side of the operator (argument). When the data type of the argument is different from that of the object itself, cast operations are executed just like a normal scalar operation.

The returned operation mode and rounding attribute are identical to those of the object itself.

PARAMETER

[I] `v` A real or integer scalar
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

4.4.19 ==

NAME

== — Compare

SYNOPSIS

```
bool operator==(const mdarray &obj) const;
```

DESCRIPTION

This operator compares the object of the mdarray class specified by the right side of the operator (argument) with the object itself. If the array size and elements of the argument `obj` are identical to those of the object itself, it returns `true`. Otherwise, it returns `false`. This member function uses the compare() function (§4.5.44) internally.

PARAMETER

[I] `obj` A object belonging to the class “mdarray”
([I] : input, [O] : output)

RETURN VALUE

true : If the sizes and values of the elements on the arrays are identical
false : If the sizes and one of the values of the elements on the arrays are not identical

EXAMPLE

The following code compares the object `my_mdarr` of the mdarray class with the object `comp_mdarr` and prints the result to stdout:

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 3);
my_mdarr = 20;

mdarray comp_mdarr(LONG_ZT);
comp_mdarr = 20;
if (my_mdarr == comp_mdarr) {
    sio.printf("true\n");
} else {
    sio.printf("false\n");
}
```

Output:

```
false
```

4.4.20 !=

NAME

!= — Compare (negative form)

SYNOPSIS

```
bool operator!=(const mdarray &obj) const;
```

DESCRIPTION

This operator compares the object of mdarray (derived) class specified by the right side of the operator (argument) with the object itself. If the argument `obj` is not identical to the object itself, it returns `true`. Otherwise, it returns `false`. This member function uses the compare() function (§4.5.44) internally.

PARAMETER

[I] `obj` A object belonging to the class "mdarray"
([I] : input, [O] : output)

RETURN VALUE

true : If the sizes and one of the values of the elements on the arrays are not identical
false : If the sizes and values of the elements on the arrays are identical

EXAMPLE

The following code compares the object `my_mdarr` of the mdarray class with the object `comp_mdarr` and prints the result to stdout:

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 3);
```

```
my_mdarr = 20;

mdarray comp_mdarr(LONG_ZT);
comp_mdarr = 20;
if (my_mdarr != comp_mdarr) {
    sio.printf("true\n");
} else {
    sio.printf("false\n");
}
```

Output:
`true`

4.5 Member Functions

4.5.1 size_type()

NAME
size_type() — Integer representing a data type (type-specific)

SYNOPSIS
`ssize_t size_type() const;`

DESCRIPTION
This member function returns an integer number corresponding to the data type. The data type of arrays in the object itself is identified by the number.

When the information of the data type is used in the code, use the constant identifier in Table 4, not a raw number such as `-4`.

RETURN VALUE
An integer representing a type of the object

EXAMPLE
The following code creates the object `my_mdarr` as the int32_t array and prints the size_type of `my_mdarr` to stdout:

```
stdstreamio sio;

mdarray my_mdarr(INT32_ZT);
sio.printf("*** my_mdarr size_type... [%zd]\n", my_mdarr.size_type());
```

Output:
`*** my_mdarr size_type... [4]`

4.5.2 bytes()

NAME
bytes() — Number of bytes of an element

SYNOPSIS
`size_t bytes() const;`

DESCRIPTION
This member function returns the byte size of an element in the array of the object itself.

RETURN VALUE
A byte length of one element

EXAMPLE
The following code creates the object `my_mdarr` as a double-precision floating-point array and prints the byte size of an element in the array to stdout:

```
stdstreamio sio;

mdarray my_mdarr(DOUBLE_ZT);
sio.printf("*** my_mdarr bytes... [%zu]\n", my_mdarr.bytes());
```

Output:
```
*** my_mdarr bytes... [8]
```

4.5.3 dim_length()

NAME
dim_length() — Number of dimensions of an array

SYNOPSIS
```
size_t dim_length() const;
```

DESCRIPTION
This member function returns the number of dimensions for the array of the object itself.

RETURN VALUE
The number of dimensions of the array

EXAMPLE
The following code prints the number of dimensions for the array of the object `my_mdarr3dim` to stdout:

```
stdstreamio sio;

mdarray my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim dim... [%zu]\n", my_mdarr3dim.dim_length());
```

Output:
```
*** my_mdarr3dim dim... [3]
```

4.5.4 length()

NAME
length() — Number of elements

SYNOPSIS
```
size_t length() const;  ..... 1
size_t length( size_t dim_index ) const;  ..... 2
```

DESCRIPTION

This member function returns the total number of elements in the arrays of the object itself. When the argument is not specified, the number of elements in dimension 1 × in dim. 2 × in dim. 3 ... is returned. When `dim_index` is passed to the argument, the number of elements in the dimension with the index `dim_index` is returned. `dim_index` starts from `0`.

PARAMETER

[I] `dim_index` The integer number (≥ 0) that specifies one of the dimensions of the array ([I] : input, [O] : output)

RETURN VALUE

The number of elements

EXAMPLE

The following code prints the total number of elements in the object `my_mdarr3dim` and the number of elements in the dimension with the index 0 to stdout:

```
stdstreamio sio;

mdarray my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim length... [%zu]\n", my_mdarr3dim.length());
sio.printf("*** my_mdarr3dim length 1dim... [%zu]\n", my_mdarr3dim.length(0));
```

Output:

```
*** my_mdarr3dim length... [60]
*** my_mdarr3dim length 1dim... [3]
```

4.5.5 byte_length()

NAME

byte_length() — Total byte size of elements (in a dimension) in an array

SYNOPSIS

```
size_t byte_length() const;  ..... 1
size_t byte_length( size_t dim_index ) const;  ..... 2
```

DESCRIPTION

This member function returns the total byte size of arrays in the object itself. When `dim_index` is passed to the argument, the byte size of an array for the dimension with the index `dim_index` is returned.

PARAMETER

[I] `dim_index` The integer number (≥0) that specifies one of the dimensions of the array ([I] : input, [O] : output)

RETURN VALUE

The whole byte size of the array or the byte size of the elements in the specified dimension

EXAMPLE

The following code prints the total byte size of the arrays in a three-dimension array `my_mdarr3dim` and the byte size of an array for the third dimension (dimension index: 2) to stdout:

```
stdstreamio sio;

mdarray my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim byte_length... [%zu]\n",
           my_mdarr3dim.byte_length());
sio.printf("*** my_mdarr3dim byte_length 3dim... [%zu]\n",
           my_mdarr3dim.byte_length(2));
```

Output:

```
*** mdarr3dim byte_length... [240]
*** mdarr3dim byte_length 3dim... [20]
```

4.5.6 col_length()

NAME

length() — The length of the array's column

SYNOPSIS

```
size_t col_length() const;
```

DESCRIPTION

This member function returns the length of the columns for the array of the object itself.

RETURN VALUE

A column length of the array

EXAMPLE

The following code prints the length of the columns for a three-dimension array `my_mdarr3dim` to stdout:

```
stdstreamio sio;

mdarray my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim col... [%zu]\n", my_mdarr3dim.col_length());
```

Output:

```
*** my_mdarr3dim col... [3]
```

4.5.7 row_length()

NAME

row_length() — The length of the array's row

SYNOPSIS

```
size_t row_length() const;
```

DESCRIPTION

This member function returns the length of the rows for the array of the object itself.

RETURN VALUE

A row length of the array

EXAMPLE

The following code prints the length of the rows for a three-dimension array `my_mdarr3dim` to stdout:

```
stdstreamio sio;

mdarray my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim row... [%zu]\n", my_mdarr3dim.row_length());
```

Output:

```
*** my_mdarr3dim row... [4]
```

4.5.8 layer_length()

NAME

layer_length() — The number of the layers of the array

SYNOPSIS

```
size_t layer_length() const;
```

DESCRIPTION

This member function returns the length of the layers for the array of the object itself. When the dimension of the array is 1 or 2, 1 is returned. When the dimension of the array is 3 or more, after the array dimension is reduced to three, the length of the layers for the third dimension (dimension index: 2) is returned.

RETURN VALUE

The number of dimensions of the array

EXAMPLE

The following code prints the length of the layers for a three-dimension array `my_mdarr3dim` to stdout:

```
stdstreamio sio;

mdarray my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim layer... [%zu]\n", my_mdarr3dim.layer_length());
```

Output:

```
*** my_mdarr3dim layer... [5]
```

4.5.9 f(), d(), ld(), c(), s(), i(), l(), ll(), i16(), i32(), i64(), z(), sz(), b(), p()

NAME

f(), d(), ld(), etc. — A reference to the specified value of the element

SYNOPSIS

```
float &f( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..................................  1
double &d( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  .................................  2
```

```
long double &ld( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..................................  3
unsigned char &c( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..................................  4
short &s( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..................................  5
int &i( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..................................  6
long &l( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..................................  7
long long &ll( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..................................  8
int16_t &i16( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..................................  9
int32_t &i32( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  .................................. 10
int64_t &i64( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  .................................. 11
size_t &z( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  .................................. 12
ssize_t &sz( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  .................................. 13
bool &b( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  .................................. 14
uintptr_t &p( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  .................................. 15
```

For each member function, the alternative function with the `const` attribute is available. They work in the same manner as the member functions in §4.5.10.

DESCRIPTION

This member function sets/gets the element specified by the arguments `idx0`, `idx1`, and `idx2` to/from the array. Different member functions are used depending on the data type of the array in the object itself.

When reading/writing a value with the member function 1 in the Automatic Resize Mode, the size of an array is resized according to the specified index. In the Manual Resize Mode, the substitution of a value into an element beyond the array size does not cause any error. The operation is ignored. In order to substitute a value into an element beyond the array size, the array size has to be extended by a member function (e.g. resize()) in advance. For more information about resize(), see §4.5.33.

When reading the element beyond the array size in Manual Resize Mode, `NAN` is returned for floating-point numbers, and any one of `INDEF_UCHAR`, `INDEF_INT16`, `INDEF_INT32`, or `INDEF_INT64` is returned for integer numbers according to the data type of the element, respectively. The value of each `INDEF` is the minimum integer value of the data type.

For the function at(), whether the member function 1 or 2 is used depends on whether the object has the "const " attribute. The member function 1 is used for the object without the "const " attribute, and the function 2 is used with the attribute automatically.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I] `idx0` Subscript for the first dimension being designated as 0
[I] `idx1` Subscript for the second dimension being designated as 1 (optional)
[I] `idx2` Subscript for the third dimension being designated as 2 (optional)
([I] : input, [O] : output)

RETURN VALUE

A reference to the value of the element according to the called member function

EXCEPTION

These member functions throw an exception when they fail to allocate a local buffer in the Automatic Resize Mode, or the byte-size of each element in the array object is smaller than that of the return value of the functions.

EXAMPLE

The following code sets values to the elements of the object `my_fmdarr` with a floating-point number array via the member function f() and prints the values of elements to stdout by f():

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(1) = 2000.2;
for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%5.1f]\n", i, my_fmdarr.f(i));
}
```

Output:

```
my_fmdarr value[0]...  [1000.1]
my_fmdarr value[1]...  [2000.2]
```

4.5.10 f_cs(), d_cs(), ld_cs(), c_cs(), s_cs(), i_cs(), l_cs(), ll_cs(), i16_cs(), i32_cs(), i64_cs(), z_cs(), sz_cs(), b_cs(), p_cs()

NAME

f_cs(), d_cs(), ld_cs(), etc. — Read one specified value of the element

SYNOPSIS

```
const float &f_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 1
const double &d_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                    ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 2
const long double &ld_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 3
const unsigned char &c_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                    ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 4
const short &s_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 5
const int &i_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                    ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 6
const long &l_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 7
```

For the object with "`const`" attribution, "`_cs`" in the member function name can be omitted.

DESCRIPTION

This member function gets the element specified by the arguments `idx0`, `idx1`, and `idx2` from the array. This function cannot set the element to the array.

When reading the element beyond the array size, `NAN` is returned for floating-point numbers, and any one of `INDEF_UCHAR`, `INDEF_INT16`, `INDEF_INT32`, or `INDEF_INT64` is returned for integer numbers according to the data type of the element, respectively. The value of each `INDEF` is the minimum integer value of the data type.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I] `idx0` Subscript for the first dimension being designated as 0
[I] `idx1` Subscript for the second dimension being designated as 1 (optional)
[I] `idx2` Subscript for the third dimension being designated as 2 (optional)
([I] : input, [O] : output)

RETURN VALUE

A reference to the value of the element according to the called member function

EXCEPTION

The member functions throw an exception when the byte-size of each element in the array object is smaller than that of the return value of the functions.

EXAMPLE

The following code sets values to the object `my_llmdarr` with a long long array and prints the values to stdout.

```
:
    stdstreamio sio;

    mdarray my_llmdarr(LLONG_ZT);
    my_llmdarr.ll(0) = 60000;
    my_llmdarr.ll(1) = 70000000;
    for ( size_t i = 0 ; i < my_llmdarr.length() ; i++ ) {
        sio.printf("my_llmdarr value[%zu]... [%lld]\n", i, my_llmdarr.ll_cs(i));
```

```
}
```

Output:

```
my_llmdarr value[0]...  [60000]
my_llmdarr value[1]...  [70000000]
```

4.5.11 dvalue()

NAME

dvalue() — The value of the element converted into the double type

SYNOPSIS

```
double dvalue( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
               ssize_t idx2 = MDARRAY_INDEF ) const;
```

DESCRIPTION

This member function casts the element in the array of the object itself to a double-precision floating-point value and returns it.

When the specified index exceeds the array size, `NAN` is returned.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I] `idx0` Subscript for the first dimension being designated as 0
[I] `idx1` Subscript for the second dimension being designated as 1 (optional)
[I] `idx2` Subscript for the third dimension being designated as 2 (optional)
([I] : input, [O] : output)

RETURN VALUE

A value converted to the double type	:	Normal end
`NAN`(error)	:	When the type of the element is not supported When the index of the elements exceeding the size of the array is specified

EXAMPLE

The following code sets a value to the mdarray_float-class object `my_mdarry` and gets the value as a double-precision floating-point number, and prints it to stdout:

```
stdstreamio sio;

mdarray_float my_mdarry;
my_mdarry[0] = 123.456;
sio.printf("my_mdarry dvalue... [\%6.3f]\n", my_mdarry.dvalue(0));
```

Output:

```
my_mdarry dvalue...  [123.456]
```

4.5.12 lvalue(), llvalue()

NAME

lvalue(), llvalue() — The value of the element converted into the long or long long type

SYNOPSIS

```
long lvalue( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
             ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 1
long long llvalue( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 2
```

DESCRIPTION

This member function casts the element in the array of the object itself to a long or long long value and returns it.

When the data type of the array is floating-point, the value is truncated to the whole number by default. In order to round it to the whole number, it requires the use of the set_rounding() function in advance. For more information about set_rounding(), see §4.5.15.

When the specified index exceeds the array size, `INDEF_LONG` or `INDEF_LLONG` is returned, respectively. The value of each `INDEF` is the minimum integer value of the data type.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I] `idx0` Subscript for the first dimension being designated as 0
[I] `idx1` Subscript for the second dimension being designated as 1 (optional)
[I] `idx2` Subscript for the third dimension being designated as 2 (optional)
([I] : input, [O] : output)

RETURN VALUE

A value converted to the long or long long type	:	Normal end
`INDEF_LONG` or `INDEF_LLONG`	:	When the type of the element is not supported When the index of the elemnts exceeding the size of the array is specified

EXAMPLE

The following code sets a value to the object `my_mdarry` with a float array and gets the value as a long number and as a long long number, and prints them to stdout:

```
stdstreamio sio;

mdarray my_mdarry(FLOAT_ZT);
my_mdarry.f(0) = 123.556;
sio.printf("my_mdarry lvalue... [%ld]\n", my_mdarry.lvalue(0));
my_mdarry.set_rounding(true);
sio.printf("my_mdarry llvalue... [%lld]\n", my_mdarry.llvalue(0));
```

Output:

```
my_mdarry lvalue...  [123]
my_mdarry llvalue...  [124]
```

4.5.13 default_value_ptr(), assign_default()

NAME

default_value_ptr(), assign_default() — Acquire and set the initial value upon size expansion

SYNOPSIS

```
const void *default_value_ptr() const;  ..... 1
mdarray &assign_default( const void *value_ptr );  ..... 2
mdarray &assign_default( double value );  ..... 3
```

DESCRIPTION

The member function 1 returns the initial value for size extension. When the value is not set up yet, `NULL` is returned.

The member functions 2 and 3 set up the initial value for size extension. The value does not apply to the existing elements. When the array size is extended, it is applied to the array.

RETURN VALUE

The member function 1 returns an address including the default value in expanding the sizes (or NULL when the default value is not defined).

The member functions 2 and 3 return a reference to itself.

EXCEPTION

The functions 2 and 3 throw an exception when they fail to allocate a local buffer.

EXAMPLE

The following code prints the initial value of the object `my_mdarr` with a float array for size extension to stdout:

```
stdstreamio sio;

mdarray my_mdarr;
my_mdarr.init(FLOAT_ZT);
my_mdarr.assign_default(50);
sio.printf("*** my_mdarr defval... [%f]\n",
           *((const float *)my_mdarr.default_value_ptr()));
```

Output:

```
*** my_mdarr defval... [50.000000]
```

4.5.14 auto_resize(), set_auto_resize()

NAME

auto_resize(), set_auto_resize() — Acquire and set the resize mode

SYNOPSIS

```
bool auto_resize() const;  ..... 1
mdarray &set_auto_resize( bool tf );  ..... 2
```

DESCRIPTION

The member function 1 returns the resize mode by boolean type. The member function 2 sets up the resize mode by boolean type.

The Automatic Resize Mode corresponds to true (=1) and the Manual Resize Mode corresponds to false (=0).

See the Operation Mode Support box in Table 6 to find out which member functions support the operation modes.

RETURN VALUE

The member function 1 returns an operation mode (or true in the Automatic Resize Mode).

The member function 2 returns a reference to itself.

EXAMPLE

The following code creates `mdarr0dim` in the Automatic Resize Mode and creates `mdarr3dim` in the Manual Resize Mode, and prints the resize modes of the arrays to stdout:

```
stdstreamio sio;

mdarray my_mdarr0dim;
sio.printf("*** my_mdarr0dim auto_resize... [%d]\n",
            (int)(my_mdarr0dim.auto_resize()));
mdarray my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim auto_resize... [%d]\n",
            (int)(my_mdarr3dim.auto_resize()));
```

Output:

```
*** my_mdarr0dim auto\_resize... [1]
*** my_mdarr3dim auto\_resize... [0]
```

4.5.15 rounding(), set_rounding()

NAME

rounding(), set_rounding() — Acquire and set the rounding off possibility

SYNOPSIS

```
bool rounding() const;  ..... 1
mdarray &set_rounding( bool tf );  ..... 2
```

DESCRIPTION

The member function 2 sets up the rounding mode. Either the floating-point numbers are truncated or rounded to the whole number in some high-level member functions. The member function 1 returns true in the round mode and false in the truncate mode.

Upon creation of the mdarray class object, it is set to the truncate mode.

When objects are copied by the operator "`=`" or the init() member function, the rounding attribution is also copied. For more information about init(), see §4.5.24.

The member functions that support the rounding attribution are: lvalue(), llvalue() (§4.5.12), assign_default() (§4.5.13), assign() (§4.5.25), convert()(§4.5.38), and all member functions for images.

RETURN VALUE

The member function 1 returns an attribute of the operation on rounding.

The member function 2 returns a reference to itself.

EXAMPLE

The following code creates the object `my_mdarr` with a long long array and sets a real number twice before and after setting the rounding mode. Then the code prints the substituted values to stdout for confirmation.

```
stdstreamio sio;

mdarray my_mdarr(LLONG_ZT);
my_mdarr.assign(1.618, 0);
sio.printf("my_mdarr value[0]... [%lld]\n", my_mdarr.ll(0));

my_mdarr.set_rounding(true);
my_mdarr.assign(1.618, 1);
sio.printf("my_mdarr value[1]... [%lld]\n", my_mdarr.ll(1));
```

Output:

```
my_imdarr value[0]...  [1]
my_imdarr value[1]...  [2]
```

4.5.16 dprint()

NAME

dprint() — Output of the object information to the standarderr output (for user's debug)

SYNOPSIS

```
void dprint() const;
```

DESCRIPTION

This member function prints the information of the object itself to stderr.

This is a function for debugging a user program.

EXAMPLE

The following code prints the information on the object `my_array` to stderr. The address of the object in `[]` is system-dependent.

```
mdarray my_array(INT_ZT, 3,2,1);
my_array.i(2,0,0) = 100;
my_array.i(0,1,0) = 200;
my_array.dprint();
```

Output:

```
sli::mdarray[obj=0x7fbffff630, sz_type=4, dim=(3,2,1)] = {
 { { 0,0,100 },
   { 200,0,0 } }
}
```

4.5.17 data_ptr()

NAME

data_ptr() — Acquire the specified element's address

SYNOPSIS

```
void *data_ptr();  ..................................................... 1
void *data_ptr( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                ssize_t idx2 = MDARRAY_INDEF );  ....................... 2
const void *data_ptr() const;  ......................................... 3
```

```
const void *data_ptr( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                      ssize_t idx2 = MDARRAY_INDEF ) const;  ..............  4
const void *data_ptr_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                         ssize_t idx2 = MDARRAY_INDEF ) const;  ...........  5
const void *data_ptr_cs() const;  ..........................................  6
```

DESCRIPTION

These member functions get the address of the element specified by the index in the array of the object itself. Member functions 3 to 6 get the address for read only.

For the function data_ptr(), whether the member functions 1, 2 or 3, 4 are used depends on whether the oobject has the "const " attribute. The member function 1 or 2 is used for the object without the "const " attribute, and the function 3 or 4 is used with the attribute automatically.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I] `idx0` Subscript for the first dimension being designated as 0
[I] `idx1` Subscript for the second dimension being designated as 1 (optional)
[I] `idx2` Subscript for the third dimension being designated as 2 (optional)
([I] : input, [O] : output)

RETURN VALUE

An address of the specified element

EXAMPLE

The following code gets the address of the value at (0, 1) in the object `my_fmdarr` with a two-dimension array and prints the value to stdout:

```
stdstreamio sio;
mdarray my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;
my_fmdarr.f(1,0) = 2000;
my_fmdarr.f(0,1) = 3000;
my_fmdarr.f(1,1) = 4000;

const float *mycarray_ptr = (float *)my_fmdarr.data_ptr_cs(0, 1);
sio.printf("*** my_fmdarr carray[0] ---> [%f] *** \n",  mycarray_ptr[0]);
```

Output:

```
*** my_fmdarr carray[0] ---> [3000.000000] ***
```

4.5.18 register_extptr()

NAME

`register_extptr()` — Register a user pointer variable

SYNOPSIS

```
mdarray &register_extptr(void *extptr_ptr);
```

DESCRIPTION

These member functions register the pointer of a user's variable to the object itself.

The initial address of the buffer managed by the object can be held in the pointer of a user's variable by this function.

Do not forget to put the operator "`&`" on the pointer variable, which is passed to the argument. (sees EXAMPLE below.)

PARAMETER

[I] `extptr_ptr` Address of the user's pointer
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code registers the pointer of `tokyoStock_ptr` to the object `tokyoStock_buf` and allocates memory for the buffer. It enables the direct access to the buffer by `tokyoStock_ptr`. Then the code prints the values of `tokyoStock_buf` to stdout for confirmation.

```
stdstreamio sio;

struct tokyoStock {
    short id;
    long stock;
};
struct tokyoStock *tokyoStock_ptr;
mdarray tokyoStock_buf(sizeof(struct tokyoStock));

tokyoStock_buf.register_extptr(&tokyoStock_ptr);
                                /* the operator "&" is needed */
tokyoStock_buf.resize(6);

tokyoStock_ptr[0].id = 3407;
tokyoStock_ptr[0].stock = 438;
tokyoStock_ptr[1].id = 4951;
tokyoStock_ptr[1].stock = 1058;
for ( size_t i = 0 ; i < tokyoStock_buf.length(0) ; i++ ) {
    sio.printf("tokyoStock_buf stock[%zu]... [%ld]\n", i,
                tokyoStock_ptr[i].stock);
}
```

Output:

```
tokyoStock_buf stock[0]...  [438]
tokyoStock_buf stock[1]...  [1058]
```

4.5.19 get_elements()

NAME

get_elements () — Copy the array itself to the user's buffer

SYNOPSIS

```
ssize_t get_elements( void *dest_buf, size_t elem_size,
                      ssize_t idx0 = 0, ssize_t idx1 = MDARRAY_INDEF,
                      ssize_t idx2 = MDARRAY_INDEF ) const;
```

DESCRIPTION

This member function copies the contents of the array for the object itself to the user buffer specified by `dest_buf`. The size of the buffer `elem_size` is set by the number of elements. The source is specified by the arguments `idx0`, `idx1`, and `idx2`.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[O]	`dest_buf`	Address of user's buffer
[I]	`elem_size`	The number of elements to be copied
[I]	`idx0`	Subscript of this array for its first dimension being designated as 0 (optional)
[I]	`idx1`	Subscript of this array for its second dimension being designated as 1 (optional)
[I]	`idx2`	Subscript of this array for its third dimension being designated as 2 (optional)

([I] : input, [O] : output)

RETURN VALUE

The number of the elements copied when the user buffer length is enough.

EXCEPTION

The function throws an exception when it detects memory corruption.

EXAMPLE

The following code copies the contents of the object `my_fmdarr` with a two-dimension array to the user buffer `myfloat` and prints the values of elements in `myfloat` to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;
my_fmdarr.f(1,0) = 2000;
my_fmdarr.f(0,1) = 3000;
my_fmdarr.f(1,1) = 4000;

float myfloat[4];
my_fmdarr.get_elements((void *)myfloat, sizeof(myfloat)/sizeof(float));
for ( int i = 0 ; i < sizeof(myfloat)/sizeof(float) ; i++ ) {
    sio.printf("myfloat value[%d]... [%f]\n", i, myfloat[i]);
}
```

Output:

```
myfloat value[0]...  [1000.000000]
myfloat value[1]...  [2000.000000]
myfloat value[2]...  [3000.000000]
myfloat value[3]...  [4000.000000]
```

4.5.20 put_elements()

NAME

put_elements () — Copy the array in the user's buffer to the array itself

SYNOPSIS

```
ssize_t put_elements( const void *src_buf, size_t elem_size, ssize_t idx0 = 0,
                      ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF );
```

DESCRIPTION

This member function copies the contents of the user buffer specified by `src_buf` to the array for the object itself. The size of the buffer `elem_size` is set by the number of elements. The destination is specified by the arguments `idx0`, `idx1`, and `idx2`.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I]	`src_buf`	Address of the user's buffer
[I]	`elem_size`	Number of elements to be copied
[I]	`idx0`	Subscript of this array for its first dimension being designated as 0 (optional)
[I]	`idx1`	Subscript of this array for its second dimension being designated as 1 (optional)
[I]	`idx2`	Subscript of this array for its third dimension being designated as 2 (optional)

([I] : input, [O] : output)

RETURN VALUE

The number of the elements copied when the user buffer length is enough

EXCEPTION

The function throws an exception when it detects memory corruption.

EXAMPLE

The following code copies the contents of the user buffer `my_float` to the object `my_fmdarr` with a two-dimension array and prints the values of elements in `my_fmdarr` to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);

float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.put_elements((const void *)my_float, sizeof(my_float)/sizeof(float));

for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
  for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                i, j, my_fmdarr.f(i, j));
  }
}
```

Output:

```
my_fmdarr value(0,0)...  [1000.000000]
```

```
my_fmdarr value(1,0)...  [2000.000000]
my_fmdarr value(0,1)...  [3000.000000]
my_fmdarr value(1,1)...  [4000.000000]
```

4.5.21 getdata()

NAME

getdata() — Copy the array itself to the user's buffer

SYNOPSIS

```
ssize_t getdata( void *dest_buf, size_t buf_size, ssize_t idx0 = 0,
                 ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF ) const;
```

DESCRIPTION

This member function copies the contents of the array for the object itself to the user buffer specified by `dest_buf`. The size of the buffer `buf_size` is set by the byte unit. The source is specified by the arguments `idx0`, `idx1`, and `idx2`.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[O]	`dest_buf`	Address of user's buffer
[I]	`buf_size`	Size of the buffer in bytes
[I]	`idx0`	Subscript of this array for its first dimension being designated as 0 (optional)
[I]	`idx1`	Subscript of this array for its second dimension being designated as 1 (optional)
[I]	`idx2`	Subscript of this array for its third dimension being designated as 2 (optional)

([I] : input, [O] : output)

RETURN VALUE

A size of the buffer copied when the user buffer size (buf_size) is enough.

EXCEPTION

The function throws an exception when memory corruption is detected.

EXAMPLE

The following code copies the contents of the object `my_fmdarr` with a two-dimension array to the user buffer `myfloat` and prints the values of elements in `myfloat` to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;
my_fmdarr.f(1,0) = 2000;
my_fmdarr.f(0,1) = 3000;
my_fmdarr.f(1,1) = 4000;

float myfloat[4];
my_fmdarr.getdata((void *)myfloat, sizeof(myfloat));
for ( int i = 0 ; i < sizeof(myfloat)/sizeof(float) ; i++ ) {
```

```
    sio.printf("myfloat value[%d]... [%f]\n", i, myfloat[i]);
}
```

Output:

```
myfloat value[0]...  [1000.000000]
myfloat value[1]...  [2000.000000]
myfloat value[2]...  [3000.000000]
myfloat value[3]...  [4000.000000]
```

4.5.22 putdata()

NAME

putdata() — Copy the array in the user's buffer to the array itself

SYNOPSIS

```
ssize_t putdata( const void *src_buf, size_t buf_size, ssize_t idx0 = 0,
                 ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF );
```

DESCRIPTION

This member function copies the contents of the user buffer specified by `src_buf` to the array for the object itself. The size of the buffer `buf_size` is set by the byte unit. The destination is specified by the arguments `idx0`, `idx1`, and `idx2`.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I]	`src_buf`	Address of user's buffer
[I]	`buf_size`	Size of the user's buffer in bytes
[I]	`idx0`	Subscript of this array for its first dimension being designated as 0 (optional)
[I]	`idx1`	Subscript of this array for its second dimension being designated as 1 (optional)
[I]	`idx2`	Subscript of this array for its third dimension being designated as 2 (optional)

([I] : input, [O] : output)

RETURN VALUE

A size of the buffer copied when the user buffer size (buf_size) is enough

EXCEPTION

The function throws an exception when it detects memory corruption.

EXAMPLE

The following code copies the contents of the user buffer `my_float` to the object `my_fmdarr` with a two-dimension array and prints the values of elements in `my_fmdarr` to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);

float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));
```

```
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
  for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
               i, j, my_fmdarr.f(i, j));
  }
}
```

Output:

```
my_fmdarr value(0,0)...  [1000.000000]
my_fmdarr value(1,0)...  [2000.000000]
my_fmdarr value(0,1)...  [3000.000000]
my_fmdarr value(1,1)...  [4000.000000]
```

4.5.23 reverse_endian()

NAME

reverse_endian() — Reverse endian if necessary

SYNOPSIS

```
mdarray_type &reverse_endian( bool is_little_endian, ssize_t sz_type = 0 );
```

DESCRIPTION

This member function is called to save the array of the object itself as a binary data in a file or load a binary data in a file to the array of the object itself.

To save data in a file, convert the endian of the data to the appropriate form for storing by this member function. Next, write the content by the stream writing function with the address obtained from the function such as data_ptr()(§4.5.17). Then call this function again for turning back the endian.

To load data from a file, read the content by the stream reading function with the address obtained from the functions such as data_ptr()(§4.5.17). Then convert the endian to the appropriate form for the system by this member function.

In both cases shown above, if the data to be stored on a file is big endian, the first argument is set to `false` (if little endian, set to `true`).

For this member function, users do not have to be conscious of the difference in system architecture. For instance, a user specifies `false` to the argument `is_little_endian` and calls this function so that a data in big endian is saved in a file. If the machine is a big endian system, the inversion process is not executed in practice. (With a little endian system, the inversion process is executed.) Next, the user saves binary data in the object in a file in a straightforward way, and the binary file in the specified byte order is created. And then, the user calls this member function with the same arguments again in order to turn back the endian if it was inverted.

This means that to save in a file this member function must be called twice with the same arguments.

The endian inversion of a data type in the byte unit can be executed by specifying the constant identifier (see the table in §4) of the data type to `sz_type`.

PARAMETER

[I]	`is_little_endian`	True when the ordering of data in a computer's memory should be little endian after one conversion
[I]	`sz_type`	Type specifier (optional)

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code outputs the binary file that contains the data of the object `my_mdarr` with a two-dimension array in big endian:

```
stdstreamio sio;

mdarray my_mdarr(INT_ZT, 2,2);
my_mdarr.i(0,0) = 10;
my_mdarr.i(1,0) = 20;
my_mdarr.i(0,1) = 30;
my_mdarr.i(1,1) = 40;

my_mdarr.reverse_endian(false);
const void *mydata_ptr = my_mdarr.data_ptr();

if ( fio.openf("w", "%s", "binary.dat") < 0 ) {
  // Error Handling
}
if ( fio.write(mydata_ptr, my_mdarr.byte_length()) < 0 ) {
  // Error Handling
}
my_mdarr.reverse_endian(false);

fio.close();
```

Output:

Contents of `binary.dat`:

```
"00 00 00 0A"
"00 00 00 14"
"00 00 00 1E"
"00 00 00 28"
```

4.5.24 init()

NAME

init() — Initialize the array

SYNOPSIS

DESCRIPTION

This member function initializes the array of the object itself.

The member functions 1 and 2 initialize the objects in the Auto matic ResizeMode. The member functions 3, 4, 5, and 6 initialize the objects in the Manual Resize Mode. The member function 7 copies all the contents and attributes of `obj` to the object itself.

The member function 1 initializes the object with the array size 0 and does not change the data type.

For the member function 2, the data type of the array is specified by `sz_type`.

For the member function 3, the data type of the array is specified by `sz_type` and the number of dimensions is specified by `ndim` and the number of elements in each dimension is specified by `naxisx[]`.

For creating the object with a one-dimension array by the member function 4, the data type of the array is specified by `sz_type` and the number of elements in the first dimension is specified by `naxis0`.

For creating the object with a two-dimension array by the member function 5, the data type of the array is specified by `sz_type` and the number of elements in the first dimension is specified by `naxis0` and the number of elements in the second dimension is specified by `naxis1`.

For creating the object with a three-dimension array by the member function 5, the data type of the array is specified by `sz_type` and the number of elements in the first dimension is specified by `naxis0` and the number of elements in the second dimension is specified by `naxis1` and the number of elements in the third dimension is specified by `naxis2`.

See Table 4 for the values that are passed to the argument `sz_type`.

PARAMETER

[I]	`sz_type`	A type specifier for each element of the array
[I]	`ndim`	Number of dimensions of the array
[I]	`naxisx[]`	Number of elements along each dimension
[I]	`naxis0`	Number of elements along the first dimension being designated as 0
[I]	`naxis1`	Number of elements along the second dimension being designated as 1
[I]	`naxis2`	Number of elements along the third dimension being designated as 2
[I]	`obj`	A reference to an input array whose contents should be copied to this object

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code initializes the object `my_mdarr` with the 2×3 integer array and prints the lengths of dimensions to stdout:

```
stdstreamio sio;

mdarray my_mdarr;
my_mdarr.init(INT_ZT, 2,3);
sio.printf("*** my_mdarr 0 dim length ====> [%zu] *** \n",
            my_mdarr.length(0));
sio.printf("*** my_mdarr 1st dim length ====> [%zu] *** \n",
            my_mdarr.length(1));
```

Output:

```
*** my_mdarr 0 dim length ====> [2] ***
*** my_mdarr 1st dim length ====> [3] ***
```

4.5.25 assign()

NAME

assign() — Substitute a value for an element(high level)

SYNOPSIS

```
mdarray &assign( double value, ssize_t idx0,
                 ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF );
```

DESCRIPTION

This member function assigns a value to the element specified by `idx`*n* in the array of the object itself. When a floating-point number is assigned to an element with the data type integer, it is truncated by default. To round it, call set_rounding() in advance. For more information about set_rounding(), see §4.5.15.

In the Automatic Resize Mode, the size of the arrays are resized according to the specified element index automatically.

In the Manual Resize Mode, assigning a value to an element beyond the array size does not cause any error. The operation is ignored. In order to assign a value to an element beyond the array size, the array size has to be extended by the member function resize() in advance. For more information about resize(), see §4.5.33.

Do not specify `MDARRAY_INDEF` for an argument explicitly.

PARAMETER

[I] `value` A real scalar in double precision
[I] `idx0` Subscript for the first dimension being designated as 0
[I] `idx1` Subscript for the second dimension being designated as 1 (optional)
[I] `idx2` Subscript for the third dimension being designated as 2 (optional)

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer in the Automatic Resize Mode.

EXAMPLE

The following code sets the value to the element specified by index 1 in the object `my_mdarr` with the float array and prints the values of elements to stdout:

```
stdstreamio sio;

mdarray my_mdarr(FLOAT_ZT);
my_mdarr.assign(200.0, 1);
for ( size_t i = 0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr lvalue[%zu]... [%ld]\n", i, my_mdarr.lvalue(i));
}
```

Output:

```
my_mdarr lvalue[0]...  [0]
my_mdarr lvalue[1]...  [200]
```

4.5.26 put()

NAME

put() — Set a value to an arbitrary element's point

SYNOPSIS

```
mdarray_type &put( type value, ssize_t idx, size_t len );  ..... 1
mdarray_type &put( type value,
                   size_t dim_index, ssize_t idx, size_t len );  ..... 2
```

DESCRIPTION

This member function puts `len` straight value(s) pointed by the address `value_ptr` to the array at the element index `idx` in the object itself. The element index and the dimension index start from 0.

Any values can be set to `idx` and `len`. In the Automatic Resize Mode, if the length of the array in the object is lower than the specified argument, the array is resized automatically. The additional part in which the value is not set is filled with the default value. In the Automatic Resize Mode, the writing operation is not executed for the elements beyond the array size.

The member function 1 writes one value pointed by the address `value_ptr` to elements `len` times in a row from `idx` in the array.

The member function 2 writes one value by the address `value_ptr` to elements `len` times in a row from `idx` in the dimension index `dim_index` of the arrays in the object. When `dim_index` is 1 or over, values are written to all elements in the lower dimensions.

PARAMETER

[I]	`value_ptr`	A pointer to a scalar
[I]	`idx`	Subscript for the first element of a subarray to which the scalar is written
[I]	`len`	Number of entries to which the scalar is written
[I]	`dim_index`	An integer number specifying one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer in the Automatic Resize Mode, or it detects memory corruption.

EXAMPLE

The following code puts two values to the array elements starting with index 1 in the object `my_smdarr` with a one-dimension array and prints the values of elements to stdout:

```
stdstreamio sio;

short s12 = 12;
const void *ptr_s12 = (const void *)&s12;

mdarray my_smdarr(SHORT_ZT, 3);
my_smdarr.put(ptr_s12, 1,2);
for ( size_t i = 0 ; i < my_smdarr.length() ; i++ ) {
    sio.printf("my_smdarr value[%zu]... [%hd]\n", i, my_smdarr.s(i));
}
```

Output:

```
my_smdarr value[0]...  [0]
my_smdarr value[1]...  [12]
my_smdarr value[2]...  [12]
```

4.5.27 swap()

NAME

swap() — Replace values between elements

SYNOPSIS

```
mdarray &swap( ssize_t idx_src, size_t len, ssize_t idx_dst );  ..... 1
mdarray &swap( size_t dim_index,
               ssize_t idx_src, size_t len, ssize_t idx_dst );  ..... 2
```

DESCRIPTION

This member function swaps values in the array of the object itself.

The member function 1 swaps `len` elements from the index `idx_src` for `len` elements from the index `idx_dst`. If `idx_dst + len` exceeds the size of the array, the process is executed for up to the size of the array.

The member function 2 swaps `len` elements from the index `idx_src` in the dimension with the index `dim_index` for `len` elements from the index `idx_dst`. If `idx_dst + len` exceeds the size of the array, the process is executed for up to the size of the array.

When the area for swapping is overlapped, only the non-overlapped area in the source area is swapped.

PARAMETER

[I]	`idx_src`	Subscript specifying the first element of one of the two subarrays to be swapped with each other
[I]	`len`	Number of elements in the subarray
[I]	`idx_dst`	Subscript specifying the first element of the other subarray
[I]	`dim_index`	The integer number that specifies one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code swaps one element specified by index 0 in the second dimension (dimension index: 1) for the element with the index 1 in the object `my_cmdarr` with the unsigned char array and prints the values of elements to stdout. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {51, 52, 101, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(my_char));

my_cmdarr.swap( 1, 0, 1, 1 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu,%zu)... [%hhu]\n",
                    i, j, my_cmdarr.c(i, j));
    }
}
```

Output:

```
my_cmdarr value(0,0)...  [101]
my_cmdarr value(1,0)...  [102]
my_cmdarr value(0,1)...  [51]
my_cmdarr value(1,1)...  [52]
```

4.5.28 move()

NAME

move() — Copy values between elements

SYNOPSIS

```
mdarray &move( ssize_t idx_src, size_t len, ssize_t idx_dst,
               bool clr );  ..................................................  1
mdarray &move( size_t dim_index, ssize_t idx_src, size_t len, ssize_t idx_dst,
               bool clr );  ..................................................  2
```

DESCRIPTION

This member function moves values in the array of the object itself.

When `false` is passed to the argument `clr`, the values of the source remain. For `true`, the values of the source do not remain and they are filled with the default values. If the value exceeding the existing array size is specified to the argument `idx_dst`, the size is not changed. This operation is different from that of the member function cpy(). (See §4.5.29.)

For the member function 1, the moving operation is always applied to the elements in the first dimension (dimension index: 0). For the member function 2, the moving operation is applied to the elements in the dimension with the index `dim_index`.

PARAMETER

[I]	`idx_src`	Subscript specifying the first element of an input subarray in this object
[I]	`len`	Number of elements in the input subarray
[I]	`idx_dst`	Subscript specifying the first element of another subarray to which the input subarray is written
[I]	`clr`	True if the contents of the input subarray may be lost
[I]	`dim_index`	The integer number that specifies one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code copies values inside the array of the object `my_cmdarr` with the unsigned char array and prints the values of elements to stdout for confirmation. The values of the source are cleared.

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 3);
my_cmdarr.c(0) = 99;
my_cmdarr.c(1) = 98;
my_cmdarr.c(2) = 97;
my_cmdarr.move( 2, 1, 0, true );
for ( size_t i = 0 ; i < my_cmdarr.length() ; i++ ) {
    sio.printf("my_cmdarr value[%zu]... [%hhu]\n",
                i, my_cmdarr.c(i));
}
```

Output:

```
my_cmdarr value[0]...  [97]
my_cmdarr value[1]...  [98]
my_cmdarr value[2]...  [0]
```

4.5.29 cpy()

NAME

cpy() — Copy values between elements (with automatic expansion)

SYNOPSIS

```
mdarray &cpy( ssize_t idx_src, size_t len, ssize_t idx_dst,
              bool clr );  ..................................................... 1
mdarray &cpy( size_t dim_index, ssize_t idx_src, size_t len, ssize_t idx_dst,
              bool clr );  ..................................................... 2
```

DESCRIPTION

This member function moves values in the array of the object itself.

When `false` is passed to the argument `clr`, the values of the source remain. For `true`, the values of the source do not remain and they are filled with the default values. If the value exceeding the existing array size is specified to the argument `idx_dst`, the size is not changed. This operation is different from that of the member function cpy(). (See §4.5.29.)

For the member function 1, the moving operation is always applied to the elements in the first dimension (dimension index: 0). For the member function 2, the moving operation is applied to the elements in the dimension with the index `dim_index`.

PARAMETER

[I]	`idx_src`	Subscript specifying the first element of an input subarray in this object
[I]	`len`	Number of elements in the input subarray
[I]	`idx_dst`	Subscript specifying the first element of another subarray to which the input subarray is written
[I]	`clr`	True if the contents of the input subarray may be lost
[I]	`dim_index`	The integer number that specifies one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code copies values inside the array of the object `my_lmdarr` with the long long array and prints the values of elements to stdout for confirmation. The values of the source remain.

```
stdstreamio sio;

mdarray my_lmdarr(LLONG_ZT);
my_lmdarr.ll(0) = -2147483646;
my_lmdarr.ll(1) = 2147483647;
my_lmdarr.cpy(1, 1, 2, false);
for ( size_t i = 0 ; i < my_lmdarr.length(0) ; i++ ) {
    sio.printf("my_lmdarr value[%zu]... [%lld]\n", i, my_lmdarr.ll(i));
}
```

Output:

```
my_lmdarr value[0]...  [-2147483646]
my_lmdarr value[1]...  [2147483647]
my_lmdarr value[2]...  [2147483647]
```

4.5.30 insert()

NAME

insert() — Insert an element

SYNOPSIS

```
mdarray &insert( ssize_t idx, size_t len );  ................................ 1
mdarray &insert( size_t dim_index, ssize_t idx, size_t len );  ............. 2
```

DESCRIPTION

This member function inserts `len` values to the index `idx` in the array of the object itself. The values are default values.

For the member function 1, the insertion is always applied to the elements in the first dimension (dimension index: 0). For the member function 2, the insertion is applied to the elements in the dimension with the index `dim_index`.

PARAMETER

[I]	`idx`	Array subscript specifying an element before which new entries should be inserted
[I]	`len`	Number of elements to be inserted
[I]	`dim_index`	The integer number that specifies one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code inserts two default values (`0`) before the first element in the long array of the object `my_mdarr` and prints the result to stdout for confirmation:

```
stdstreamio sio;

mdarray my_mdarr(LONG_ZT, 2);
my_mdarr.l(0) = -2147483646;
my_mdarr.l(1) = 2147483647;
my_mdarr.insert( 1, 2 );
for ( size_t i = 0 ; i < my_mdarr.length(0) ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%ld]\n", i,
                my_mdarr.l(i));
}
```

Output:

```
my_mdarr value[0]...  [-2147483646]
my_mdarr value[1]...  [0]
my_mdarr value[2]...  [0]
my_mdarr value[3]...  [2147483647]
```

4.5.31 crop()

NAME

crop() — Extract an element

SYNOPSIS

```
mdarray &crop( ssize_t idx, size_t len );  ..................................  1
mdarray &crop( size_t dim_index, ssize_t idx, size_t len );  ..............  2
```

DESCRIPTION

This member function extracts `len` values from the index `idx` in the array of the object itself.

For the member function 1, the extraction is always applied to the elements in the first dimension (dimension index: 0). For the member function 2, the extraction is applied to the elements in the dimension with the index `dim_index`.

PARAMETER

[I]	`idx`	Array subscript specifying the first element to be extracted
[I]	`len`	Number of elements to be extracted
[I]	`dim_index`	The integer number that specifies one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code extracts one-dimension elements specified by index 1 in the first dimension (dimension index: 0) from the object `my_cmdarr` with the unsigned char array and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 2, 3);
my_cmdarr.c(0,0) = 124;
my_cmdarr.c(1,0) = 125;
my_cmdarr.c(0,1) = 126;
my_cmdarr.c(1,1) = 127;
my_cmdarr.crop( 0, 1, 1 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu, %zu)... [%hhu]\n",
                    i, j, my_cmdarr.c(i, j));
    }
}
```

Output:

```
my_cmdarr value(0, 0)...  [125]
my_cmdarr value(0, 1)...  [127]
my_cmdarr value(0, 2)...  [0]
```

4.5.32 erase()

NAME

erase() — Erase an element

SYNOPSIS

```
mdarray &erase( ssize_t idx, size_t len );  ..................................  1
mdarray &erase( size_t dim_index, ssize_t idx, size_t len );  ..............  2
```

DESCRIPTION

This member function erases the specified elements from the array of the object itself. The length of the array is reduced by `len`.

For the member function 1, the erasing operation is always applied to the elements in the first dimension (dimension index: 0). For the member function 2, the erasing operation is applied to the elements in the dimension with the index `dim_index`.

PARAMETER

[I]	`idx`	Array subscript specifying the first element to be removed
[I]	`len`	Number of elements to be removed
[I]	`dim_index`	The integer number that specifies one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code erases one element specified by index 1 in the object `my_mdarr` with a one-dimension array and prints the values of elements to stdout:

```
stdstreamio sio;

mdarray my_mdarr(LLONG_ZT, 3);
my_mdarr.ll(0) = 0;
my_mdarr.ll(1) = 2147483646;
my_mdarr.ll(2) = 2147483647;

my_mdarr.erase( 1, 1 );
for ( size_t i = 0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%lld]\n", i, my_mdarr.ll(i));
}
```

Output:

```
my_mdarr value[0]...  [0]
my_mdarr value[1]...  [2147483647]
```

4.5.33 resize()

NAME

resize() — Change the length of the array

SYNOPSIS

```
mdarray_type &resize( size_t len );  ..... 1
mdarray_type &resize( size_t dim_index, size_t len );  ..... 2
mdarray_type &resize( const mdarray &src );  ..... 3
```

DESCRIPTION

This member function resizes the length of the array in the object itself.

For extension, new values of the elements are filled with the default values. For reduction, elements after the index `len` are removed.

For the member function 1, the resizing operation is always applied to the elements in the first dimension (dimension index: 0). For the member function 2, the resizing operation is applied to the elements in the dimension with the index `dim_index`. For the member function 3, the number of dimensions and the length of the array are conformed to that of the object `src`.

PARAMETER

[I] `len` Number of elements after being resized

[I] `dim_index` The integer number that specifies one of the dimensions of the array

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code resizes the length of the array for the second dimension (dimension index: 1) in the object `my_cmdarr` with a two-dimension array to 3 and prints the result to stdout for confirmation:

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 2, 2);
my_cmdarr.c(0,0) = 70;
my_cmdarr.c(1,0) = 71;
my_cmdarr.c(0,1) = 36;
my_cmdarr.c(1,1) = 37;

my_cmdarr.resize( 1, 3 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu, %zu)... [%hhu]\n",
                    i, j, my_cmdarr.c(i, j));
    }
}
```

Output:

```
my_cmdarr value(0, 0)...  [70]
my_cmdarr value(1, 0)...  [71]
my_cmdarr value(0, 1)...  [36]
my_cmdarr value(1, 1)...  [37]
my_cmdarr value(0, 2)...  [0]
my_cmdarr value(1, 2)...  [0]
```

4.5.34 resizeby()

NAME

resizeby() — Change the length of the array relatively

SYNOPSIS

```
mdarray_type &resizeby( ssize_t len );  ..................................... 1
mdarray_type &resizeby( size_t dim_index, ssize_t len );  ................... 2
```

DESCRIPTION

This member function resizes the length of the array in the object itself by `len`.

For reduction, a negative value is passed to the argument `len`. The size of the resized array is the original size plus `len`.

For the member function 1, the resizing operation is always applied to the elements in the first dimension (dimension index: 0). For the member function 2, the resizing operation is applied to the elements in the dimension with the index `dim_index`.

PARAMETER

[I] `len` Number of elements to be increased or decreased
[I] `dim_index` The integer number that specifies one of the dimensions of the array
([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code resizes the length of the array in the object `my_cmdarr` with a one-dimension array and prints the values to stdout for confirmation:

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 3);

my_cmdarr.resizeby( -2 );
for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
    sio.printf("my_cmdarr value[%d]... [%hhu]\n", i, my_cmdarr.c(i));
}
```

Output:

```
my_cmdarr value[0]...  [0]
```

4.5.35 increase_dim()

NAME

increase_dim() — Expand the number of dimensions

SYNOPSIS

```
mdarray_type &increase_dim();
```

DESCRIPTION

This member function increments the dimension of the array in the object itself.

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer.

EXAMPLE

The following code increments the dimension of the array in the object `my_mdarr` with a three-dimension array and prints the number of dimensions to stdout:

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 1, 2, 3);
my_mdarr.increase_dim();
sio.printf("my_mdarr dim... [%zu]\n",  my_mdarr.dim_length());
```

Output:
```
my_mdarr dim...  [4]
```

4.5.36 decrease_dim()

NAME
decrease_dim() — Reduce the number of dimensions

SYNOPSIS
```
mdarray_type &decrease_dim();
```

DESCRIPTION
This member function decrements the dimension of the array in the object itself.

RETURN VALUE
A reference to itself

EXCEPTION
The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE
The following code decrements the dimension of the array in the object `my_mdarr` with a three-dimension array and prints the number of dimensions to stdout:

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 1, 2, 3);
my_mdarr.decrease_dim();
sio.printf("my_mdarr dim... [%zu]\n",  my_mdarr.dim_length());
```

Output:
```
my_mdarr dim...  [2]
```

4.5.37 swap()

NAME
swap() — Replace the object by another one

SYNOPSIS
```
mdarray_type &swap( mdarray_type &sobj );
```

DESCRIPTION
This member function swaps the specified object `sobj` for the object itself. All attributes such as the size of arrays are exchanged.

PARAMETER

[I/O] `sobj` Object belonging to the same class as this instance

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code swaps the object `my_fmdarr` with a one-dimension array for the object `swap_mdarr` and prints the values of elements in the `my_fmdarr` to stdout:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2);
my_fmdarr.f(0) = 1000;
my_fmdarr.f(1) = 2000;

mdarray swap_mdarr(DOUBLE_ZT, 2);
swap_mdarr.d(0) = 100;
swap_mdarr.d(1) = 200;
my_fmdarr.swap(swap_mdarr);
for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%g]\n", i,
                my_fmdarr.dvalue(i));
}
```

output:

```
my_fmdarr value[0]...  [100]
my_fmdarr value[1]...  [200]
```

4.5.38 convert()

NAME

convert() — Convert the type of the array

SYNOPSIS

```
mdarray &convert( ssize_t sz_type );  ..................................... 1
mdarray &convert( ssize_t sz_type, void (*func)(const void *,void *,void *),
                  void *user_ptr );  ..................................... 2
```

DESCRIPTION

This member function converts the data type of the array in the object itself into the data type specified by `sz_type`. Underflow or overflow occurs depending on the combination of the source data type and the destination data type.

For the member function 2, the behavior of converting is customizable by a user-defined function. The first argument of the user-defined function is the address of the original element in the array, the second is the address of the converted element, and the third is `user_ptr`.

PARAMETER

[I] `func` Address of user-defined function

[I] `user_ptr` The pointer that is given to the above function as its last argument

([I] : input, [O] : output)

RETURN VALUE
A reference to itself

EXCEPTION
The function throws an exception when it fails to allocate a local buffer.

EXAMPLE
The following code converts the data type of the object `my_fmdarr` with a one-dimension array from floating-point into integer and prints the values of elements to stdout for confirmation.

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(1) = -2000.6;
my_fmdarr.convert(INT_ZT);
for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%d]\n", i, my_fmdarr.i(i));
}
```

output:
```
my_fmdarr value[0]...  [1000]
my_fmdarr value[1]...  [-2000]
```

4.5.39 ceil()

NAME
ceil() — Raise decimals to the next whole number in a double type value

SYNOPSIS
```
mdarray_type &ceil();
```

DESCRIPTION
This member function rounds up all elements (floating-point number) of the array in the object itself to the nearest integer.

RETURN VALUE
A reference to itself

EXAMPLE
The following code sets the values with decimal places to the object `my_fmdarr` with a one-dimension array and rounds them up, and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(1) = 2000.6;
my_fmdarr.ceil();
```

```
for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr.f(i));
}
```

Output:

```
my_fmdarr value[0]...  [1001.000000]
my_fmdarr value[1]...  [2001.000000]
```

4.5.40 floor()

NAME

floor() — Devalue decimals in a double type value

SYNOPSIS

```
mdarray_type &floor();
```

DESCRIPTION

This member function rounds down all elements (floating-point number) of the array in the object itself to the nearest integer.

RETURN VALUE

A reference to itself

EXAMPLE

The following code sets the values with decimal places to the object `my_fmdarr` with a one-dimension array and rounds them down, and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(1) = 2000.9;
my_fmdarr.floor();

for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr.f(i));
}
```

Output:

```
my_fmdarr value[0]...  [1000.000000]
my_fmdarr value[1]...  [2000.000000]
```

4.5.41 round()

NAME

round() — Round off decimals in a double type value

SYNOPSIS

```
mdarray_type &round();
```

DESCRIPTION

This member function rounds all elements (floating-point number) of the array in the object itself to the nearest integer.

RETURN VALUE

A reference to itself

EXAMPLE

The following code sets the values with decimal places to the object `mdarrf` with a one-dimension array and rounds them to the nearest integer, and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);
my_fmdarr.f(0) = 1000.5;
my_fmdarr.f(1) = -1000.5;
my_fmdarr.round();

for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr.f(i));
}
```

output:

```
my_fmdarr value[0]...  [1001.000000]
my_fmdarr value[1]...  [-1001.000000]
```

4.5.42 trunc()

NAME

trunc() — Omit decimals in a double type value

SYNOPSIS

```
mdarray_type &trunc();
```

DESCRIPTION

This member function rounds down all elements (floating-point number) of the array in the object itself to the integer which is closer to 0 than the element.

RETURN VALUE

A reference to itself

EXAMPLE

The following code sets the values with decimal places to the object `my_fmdarr` with a one-dimension array and rounds them down to the integers which are closer to 0 than the elements, and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);
my_fmdarr.f(0) = 1.7;
my_fmdarr.f(1) = -1.7;
```

```
my_fmdarr.trunc();

for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr.f(i));
}
```

Output:

```
my_fmdarr value[0]...  [1.000000]
my_fmdarr value[1]...  [-1.000000]
```

4.5.43 abs()

NAME

abs() — Absolute value of all elements

SYNOPSIS

```
mdarray_type &abs();
```

DESCRIPTION

This member function returns the absolute values of all elements of the array in the object itself.

RETURN VALUE

A reference to itself

EXAMPLE

The following code sets negative values to the elements of the object `my_fmdarr` with a one-dimension array and prints the absolute values of elements to stdout:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = -1000.1;
my_fmdarr.f(1) = -2000.6;
my_fmdarr.abs();

for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%5.1f]\n", i, my_fmdarr.f(i));
}
```

Output:

```
my_fmdarr value[0]...  [1000.1]
my_fmdarr value[1]...  [2000.6]
```

4.5.44 compare()

NAME

compare() — Compare array objects

SYNOPSIS

```
bool compare(const mdarray &obj) const;
```

DESCRIPTION

This member function compares the array of the object itself with that of the specified object `obj`.

Even when the data types are not identical, this member function returns true (=1) if the length and values of the array are identical. If not, it returns false (=0).

PARAMETER

[I] `obj` Object belonging to the class "mdarray"

([I] : input, [O] : output)

RETURN VALUE

true : If the sizes and values of the elements on the arrays are identical

false : If the sizes and one of the values of the elements on the arrays are not identical

EXAMPLE

The following code compares the object `my_fmdarr` with a two-dimension array with the object `my_i64mdarr` with a two-dimension array and prints the result to stdout:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;

mdarray my_i64mdarr(INT64_ZT, 2,2);
my_i64mdarr.i64(0,0) = 1000;

sio.printf("*** my_fmdarr compare [%d] *** \n",
            (int)my_fmdarr.compare(my_i64mdarr));
```

output:

```
*** my_fmdarr compare [1] ***
```

4.5.45 copy()

NAME

copy() — Copy a part of an array into another object (for image data)

SYNOPSIS

```
ssize_t copy( mdarray *dest ) const;
ssize_t copy( mdarray &dest ) const;
```

DESCRIPTION

This member function copies all the contents of the object itself to the specified object `dest`.

All attributes such as the data type, the length, and values of the source array are copied to the destination array. This member function does not affect the array of the object itself (source).

PARAMETER

[O] `dest` an object in the class "mdarray" to which the whole contents of this array is written

([I] : input, [O] : output)

RETURN VALUE

The number of copied elements (column × row × layer)

EXCEPTION

The function throws an exception when it fails to allocate a buffer, or it detects memory corruption.

EXAMPLE

The following code copies the object `my_cmdarr` with a two-dimension array to the object `my_fmdarr` and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT);
mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {99, 101, 98, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(my_char));

ssize_t copy_size = my_cmdarr.copy( my_fmdarr );

for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%hhu]\n",
                    i, j, my_fmdarr.c(i, j));
    }
}
```

Output:

```
my_fmdarr value(0,0)...  [98]
my_fmdarr value(1,0)...  [99]
my_fmdarr value(0,1)...  [101]
my_fmdarr value(1,1)...  [102]
```

4.5.46 copy()

NAME

copy() — Copy a part of an array into another object (for image data)

SYNOPSIS

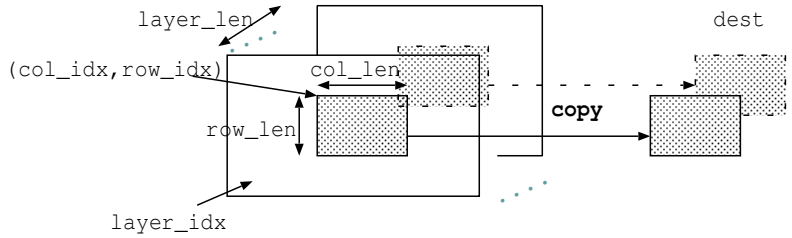
```
ssize_t copy( mdarray *dest,
              ssize_t col_idx, size_t col_len=MDARRAY_ALL,
              ssize_t row_idx=0, size_t row_len=MDARRAY_ALL,
              ssize_t layer_idx=0, size_t layer_len=MDARRAY_ALL ) const;
ssize_t copy( mdarray &dest,
              ssize_t col_idx, size_t col_len=MDARRAY_ALL,
              ssize_t row_idx=0, size_t row_len=MDARRAY_ALL,
              ssize_t layer_idx=0, size_t layer_len=MDARRAY_ALL ) const;
```

DESCRIPTION

This member function is for image data and copies a part of contents of the object itself to the specified object `dest`.

The element values and the data types of the source arrays are copied to the destination arrays. This member function does not affect the array of the object itself (source).

Image data is copied by copy() as shown below:



The shaded area in the figure is specified by the second or higher arguments, and it is copied to the `dest`.

Do not specify `MDARRAY_ALL` for an argument explicitly.

PARAMETER

[O]	`dest`	An instance of the class "mdarray_type" to which a subarray of this object should be written
[I]	`col_idx`	Subscript specifying the first column of the subarray
[I]	`col_len`	Number of columns in the subarray
[I]	`row_idx`	Subscript specifying the first row of the subarray
[I]	`row_len`	Number of rows in the subarray
[I]	`layer_idx`	Subscript specifying the first layer of the subarray
[I]	`layer_len`	Number of layers in the subarray

([I] : input, [O] : output)

RETURN VALUE

The number of copied elements (column × row × layer)

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code copies the object `my_cmdarr` with a two-dimension array to the object `my_fmdarr` and prints the values of elements to stdout for confirmation. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 2,2);
unsigned char my_char[] = {98, 99, 101, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(my_char));

mdarray my_dmdarr(DOUBLE_ZT ,2,2);
double my_d[] = {-501, 501, -502, 502};
my_dmdarr.putdata((const void *)my_d, sizeof(my_d));

ssize_t ret_size = my_cmdarr.copy( my_dmdarr, 1, 1, 1, 1 );
for ( size_t j = 0 ; j < my_dmdarr.length(1) ; j++ ) {
```

```
    for ( size_t i = 0 ; i < my_dmdarr.length(0) ; i++ ) {
        /* The data type of my_dmdarr is changed into */
        /*  unsigned char by copy. */
        sio.printf("my_dmdarr value(%zu,%zu)... [%hhu]\n",
                    i, j, my_dmdarr.c(i, j));
    }
}
```

Output:

```
my_ldmdarr value(0,0)...  [102]
```

4.5.47 cut()

NAME

cut() — Cut all values in an array and copy them into another object

SYNOPSIS

```
mdarray_type &cut( mdarray_type *dest );
mdarray_type &cut( mdarray_type &dest );
```

DESCRIPTION

This member function cuts all contents of the array of the object itself and copies it to the specified object `dest`.

Since all contents of the array of the object itself are cut, the length of the array (source) is set to 0.

PARAMETER

[O] `dest` The object to which this array is written

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a buffer, or it detects memory corruption.

EXAMPLE

The following code cuts the object `my_cmdarr` with a two-dimension array and copies it to the object `my_mdarr`, and prints the length of the array of the `my_cmdarr` to stdout for confirmation. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_mdarr;

mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {51, 101, 52, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(my_char));

my_cmdarr.cut( my_mdarr );
sio.printf("my_cmdarr length()... [%zu]\n", my_cmdarr.length());
```

Output:
```
my_cmdarr length()...  [0]
```

4.5.48 cut()

NAME

cut() — Cut a part of values in an array and copy them into another object (for image data)

SYNOPSIS

```
mdarray &cut( mdarray *dest,
              ssize_t col_idx, size_t col_len=MDARRAY_ALL,
              ssize_t row_idx=0, size_t row_len=MDARRAY_ALL,
              ssize_t layer_idx=0, size_t layer_len=MDARRAY_ALL );
mdarray &cut( mdarray &dest,
              ssize_t col_idx, size_t col_len=MDARRAY_ALL,
              ssize_t row_idx=0, size_t row_len=MDARRAY_ALL,
              ssize_t layer_idx=0, size_t layer_len=MDARRAY_ALL );
```

DESCRIPTION

This member function is for image data and cuts a part of contents of the object itself, and copies it to the specified object `dest`.

The length of the array of the object itself (source) is not changed and the values of the area specified by the second or higher arguments are filled with default values.

Do not specify `MDARRAY_ALL` for an argument explicitly.

PARAMETER

[O]	`dest`	An instance of the class "mdarray_type" to which a subarray of this object should be written
[I]	`col_idx`	Subscript specifying the first column of the subarray
[I]	`col_len`	Number of columns in the subarray
[I]	`row_idx`	Subscript specifying the first row of the subarray
[I]	`row_len`	Number of rows in the subarray
[I]	`layer_idx`	Subscript specifying the first layer of the subarray
[I]	`layer_len`	Number of layers in the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXCEPTION

The function throws an exception when it fails to allocate a local buffer, or it detects memory corruption.

EXAMPLE

The following code cuts the zeroth column of the object `my_cmdarr` with a two-dimension array and copies it to the object `my_mdarr` and prints the values of elements to stdout for confirmation. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_mdarr;
```

```
mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {51, 101, 52, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(my_char));

my_cmdarr.cut( my_mdarr, 0, 1 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu,%zu)... [%hhu]\n",
                    i, j, my_cmdarr.c(i, j));
    }
}
```

Output:

```
my_cmdarr value(0,0)...  [0]
my_cmdarr value(1,0)...  [101]
my_cmdarr value(0,1)...  [0]
my_cmdarr value(1,1)...  [102]
```

4.5.49 clean()

NAME

clean() — Padding of existing values in an array by default ones (for image data)

SYNOPSIS

```
mdarray_type &clean( ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
                     ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
                     ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );
```

DESCRIPTION

This member function fills the array elements of the object itself with default values. The arguments are optional. When no argument is specified, the cleaning operation is applied to all elements. The length of the array is not changed by `clean()`.

This member function is for image data.

Do not specify `MDARRAY_ALL` for an argument explicitly.

PARAMETER

[I]	`col_index`	Subscript specifying the first column of a subarray in this object
[I]	`col_size`	Number of columns of the subarray
[I]	`row_index`	Subscript specifying the first row of the subarray
[I]	`row_size`	Number of rows of the subarray
[I]	`layer_index`	Subscript specifying the first layer of the subarray
[I]	`layer_size`	Number of layers of the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code sets values to the elements of the object `my_smdarr` with a two-dimension array and cleans an element, and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_smdarr(SHORT_ZT, 2,2);
my_smdarr.s(0,0) = 1;
my_smdarr.s(1,0) = 3;
my_smdarr.s(0,1) = 2;
my_smdarr.s(1,1) = 4;

my_smdarr.clean(1,1,1,1);
for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
        sio.printf("my_smdarr value(%zu,%zu)... [%hd]\n",
                   i, j, my_smdarr.s(i, j));
    }
}
```

Output:

```
my_smdarr value(0,0)...  [1]
my_smdarr value(1,0)...  [3]
my_smdarr value(0,1)...  [2]
my_smdarr value(1,1)...  [0]
```

4.5.50 fill()

NAME

fill() — Rewrite element values (for image data)

SYNOPSIS

```
mdarray &fill( double value,
     ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
     ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
     ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL ); ..... 1
mdarray &fill( double value,
     double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,void *),
     void *user_ptr,
     ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
     ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
     ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );  ..... 2
mdarray &fill( double value,
     void (*func_dest2d)(const void *,void *,void *), void *user_ptr_dest2d,
     void (*func_d2dest)(const void *,void *,void *), void *user_ptr_d2dest,
     double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,void *),
     void *user_ptr_func,
     ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
     ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
     ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL ); ..... 3
```

DESCRIPTION

This member function rewrites the specified array elements of the object itself as the argument `value` (Function 1). This member function rewrites the specified array elements of the object itself via the user-defined function `func` (Function 2).

The arguments of the user-defined function are, from the left, the value of the object itself, the value specified by `value`, the index of a column, the index of a row, the index of a layer, the address of the object itself, and the user pointer `user_ptr`, respectively. To find out how to specify a user-defined function, see EXAMPLE in §4.5.53.

This member function is for image data.

Do not specify `MDARRAY_ALL` for an argument explicitly.

PARAMETER

[I]	`value`	A real scalar to be written to a subarray of this object
[I]	`user_ptr`	The user's pointer that is given to the function "func" as its last argument
[I]	`user_ptr_dest2d`	The user's pointer that is given to the function "func_dest2d" as its last argument
[I]	`user_ptr_d2dest`	The user's pointer that is given to the function "func_d2dest" as its last argument
[I]	`col_index`	Subscript specifying the first column of the subarray
[I]	`col_size`	Number of columns of the subarray
[I]	`row_index`	Subscript specifying the first row of the subarray
[I]	`row_size`	Number of rows of the subarray
[I]	`layer_index`	Subscript specifying the first layer of the subarray
[I]	`layer_size`	Number of layers of the subarray
[I]	`func`	A pointer to the user's function for data conversion
[I]	`func_dest2d`	A pointer to the user's function that converts raw data stored in this object to an array of intrinsic real numbers in double precision
[I]	`func_d2dest`	A pointer to the user's function that converts an array of intrinsic real numbers in double precision to raw data stored in this object.

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code fills all elements of the object `my_smdarr` with a two-dimension array with `100` and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_smdarr(SHORT_ZT, 2,2);
my_smdarr.fill(100);

for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
        sio.printf("my_smdarr value(%zu,%zu).. [%hd]\n",
                   i, j, my_smdarr.s(i, j));
    }
}
```

Output:

```
my_smdarr value(0,0)..  [100]
my_smdarr value(1,0)..  [100]
my_smdarr value(0,1)..  [100]
my_smdarr value(1,1)..  [100]
```

4.5.51 add()

NAME

add() — Add element values (for image data)

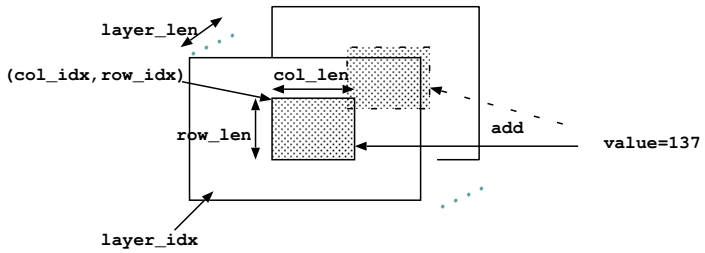
SYNOPSIS

```
mdarray_type &add( double value,
                   ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
                   ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
                   ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );
```

DESCRIPTION

This member function adds the value of the argument `value` to array elements of the object itself specified by arguments. This member function is for image data.

The value 137 is added to a part of image data by add() as shown below:



The shaded area in the figure is specified by the second or later arguments, and `value` is added to the area.

Do not specify `MDARRAY_ALL` for an argument explicitly.

PARAMETER

[I]	`value`	A real scalar to be added to a subarray of this object
[I]	`col_index`	Subscript specifying the first column of the subarray
[I]	`col_size`	Number of columns of the subarray
[I]	`row_index`	Subscript specifying the first row of the subarray
[I]	`row_size`	Number of rows of the subarray
[I]	`layer_index`	Subscript specifying the first layer of the subarray
[I]	`layer_size`	Number of layers of the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code adds `10` to the value in the second column and second row of the object `my_smdarr` with a two-dimension array and prints the values of elements to stdout for confirmation. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_smdarr(SHORT_ZT, 2,2);
short my_short[] = {1, 2, 3, 4};
```

```
my_smdarr.putdata((const void *)my_short, sizeof(my_short));
my_smdarr.add(10.0, 1,1,1,1);

for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
        sio.printf("my_smdarr value(%zu,%zu)... [%hd]\n",
                   i, j, my_smdarr.s(i, j));
    }
}
```

Output:

```
my_smdarr value(0,0)...  [1]
my_smdarr value(1,0)...  [2]
my_smdarr value(0,1)...  [3]
my_smdarr value(1,1)...  [14]
```

4.5.52 multiply()

NAME

multiply() — Multiply element values (for image data)

SYNOPSIS

```
mdarray_type &multiply( double value,
                        ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
                        ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
                        ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );
```

DESCRIPTION

This member function multiplies the specified array elements of the object itself by the value of the argument `value`. This member function is for image data.

Do not specify `MDARRAY_ALL` for an argument explicitly.

PARAMETER

[I]	`value`	A real scalar to be multiplied to a subarray of this object
[I]	`col_index`	Subscript specifying the first column of the subarray
[I]	`col_size`	Number of columns of the subarray
[I]	`row_index`	Subscript specifying the first row of the subarray
[I]	`row_size`	Number of rows of the subarray
[I]	`layer_index`	Subscript specifying the first layer of the subarray
[I]	`layer_size`	Number of layers of the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code multiplies all elements of the object `my_fmdarr` with a two-dimension array by `50` and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
```

```
float my_float[] = {1, 3, 2, 4};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

my_fmdarr.multiply(50);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                   i, j, my_fmdarr.f(i, j));
    }
}
```

Output:

```
my_fmdarr value(0,0)...  [50.000000]
my_fmdarr value(1,0)...  [150.000000]
my_fmdarr value(0,1)...  [100.000000]
my_fmdarr value(1,1)...  [200.000000]
```

4.5.53 paste()

NAME

paste() — Paste up an array object (for image data)

SYNOPSIS

```
mdarray &paste( const mdarray &src,
  ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 ); ... 1
mdarray &paste( const mdarray &src,
  double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,void *),
  void *user_ptr,
  ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 ); ... 2
mdarray &paste( const mdarray &src,
  void (*func_src2d)(const void *,void *,void *), void *user_ptr_src2d,
  void (*func_dest2d)(const void *,void *,void *), void *user_ptr_dest2d,
  void (*func_d2dest)(const void *,void *,void *), void *user_ptr_d2dest,
  double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,void *),
  void *user_ptr,
  ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 ); ... 3
```

DESCRIPTION

This member function pastes the element values specified by `src` into the specified region of the array in the object itself (Function 1). These member functions paste the element values converted via a user-defined function into the specified region of the array in the object itself (Functions 2, 3).

For the member functions 2 and 3, the behaviors of pasting are customizable by a user-defined function. The arguments of the user-defined function `func` are, from the left, the value of the object itself, the value of the object `src`, the index of a column, the index of a row, the index of a layer, the address of the object itself, and the user pointer `user_ptr`, respectively.

This member function is for image data.

PARAMETER

[I]	`src`	A object containing an input array
[I]	`func_src2d`	A pointer to the user's function that converts raw data in the object "src" to an array of intrinsic real numbers in double precision
[I]	`user_ptr_src2d`	The user's pointer that is given to the function "func_src2d" as its last argument
[I]	`func_dest2d`	A pointer to the user's function that converts raw data in this object to an array of intrinsic real numbers in double precision
[I]	`user_ptr_dest2d`	The user's pointer that is given to the function "func_dest2d" as its last argument
[I]	`func_d2dest`	A pointer to the user's function that converts an array of intrinsic real numbers in double precision to raw data stored in this object.
[I]	`user_ptr_d2dest`	The user's pointer that is given to the function "func_d2dest" as its last argument
[I]	`func`	A pointer to the user's function that defines an operation to be performed on elements of two input arrays.
[I]	`user_ptr`	The user's pointer given to the function "func" as its last argument
[I]	`dest_col`	Subscript specifying the first column of a subarray on which the input data should be pasted.
[I]	`dest_row`	Subscript specifying the first row of the subarray on which the input data should be pasted.
[I]	`dest_layer`	Subscript specifying the first layer of the subarray on which the input data should be pasted.

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code pastes the object `mypaste_mdarr` with a two-dimension array into the object `my_fmdarr` with a two-dimension array. In pasting, the values of the objects are summed up and 500 is added to the values via the user-defined function. The values of elements are printed to stdout for confirmation. For more information about `putdata()`, see §4.5.22:

```
double my_func(double self, double src, ssize_t x,
               ssize_t y, ssize_t z, mdarray *myptr, void *p)
{
    return self + src + 500;
}

/* main */
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
float my_float[] = {100, 0, 200};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

mdarray mypaste_mdarr(FLOAT_ZT, 2,2);
float mypaste_float[] = {1000, 3000, 2000, 4000};
mypaste_mdarr.putdata((const void *)mypaste_float, sizeof(mypaste_float));
```

```
my_fmdarr.paste(mypaste_mdarr, &my_func, NULL);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value[%zu][%zu]... [%f]\n", i, j,
                   my_fmdarr.f(i, j));
    }
}
```

Output:

```
my_fmdarr value(0,0)...  [1600.000000]
my_fmdarr value(1,0)...  [2500.000000]
my_fmdarr value(0,1)...  [3700.000000]
my_fmdarr value(1,1)...  [4500.000000]
```

4.5.54 add()

NAME

add() — Add an array object (for image data)

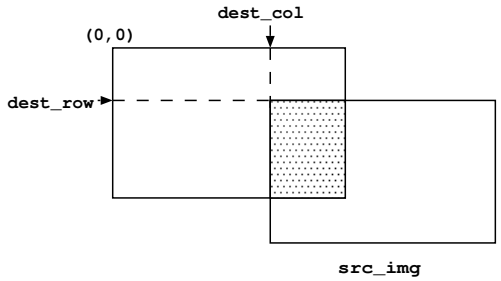
SYNOPSIS

```
mdarray &add( const mdarray &src_img, ssize_t dest_col = 0,
              ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

This member function adds array elements of the object `src_img` to those of the object itself. A start position for addition can be specified separately for columns, rows, and layers. This member function is for image data.

Image data is added to the elements of the object itself by add() as shown below:



The shaded area in the figure is specified by the second or later arguments, and `src_img` is added to the area.

PARAMETER

[I]	`src_img`	An instance of the class "mdarray" to be added to a subarray of this object
[I]	`dest_col`	Subscript specifying the first column of the subarray
[I]	`dest_row`	Subscript specifying the first row of the subarray
[I]	`dest_layer`	Subscript specifying the first layer of the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code adds the object `add_smdarr` with a two-dimension array to the object `my_smdarr` with a two-dimension array and prints the values of elements to stdout for confirmation. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_smdarr(SHORT_ZT, 2,2);
short my_short[] = {1, 2, 3, 4};
my_smdarr.putdata((const void *)my_short, sizeof(my_short));

mdarray myadd_smdarr(SHORT_ZT, 2,2);
short myadd_short[] = {9, 8, 7, 6};
myadd_smdarr.putdata((const void *)myadd_short, sizeof(myadd_short));

my_smdarr.add(myadd_smdarr);
for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
        sio.printf("my_smdarr value(%zu,%zu)... [%hd]\n",
                    i, j, my_smdarr.s(i, j));
    }
}
```

Output:

```
my_smdarr value(0,0)...  [10]
my_smdarr value(1,0)...  [10]
my_smdarr value(0,1)...  [10]
my_smdarr value(1,1)...  [10]
```

4.5.55 subtract()

NAME

subtract() — Subtract an array object (for image data)

SYNOPSIS

```
mdarray &subtract( const mdarray &src_img, ssize_t dest_col = 0,
                   ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

This member function subtracts array elements of the object `src_img` from those of the object itself. A start position for subtraction can be specified separately for columns, rows, and layers. This member function is for image data.

PARAMETER

[I]	`src_img`	An instance of the class "mdarray" to be subtracted from a subarray of this object
[I]	`dest_col`	Subscript specifying the first column of the subarray
[I]	`dest_row`	Subscript specifying the first row of the subarray
[I]	`dest_layer`	Subscript specifying the first layer of the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code subtracts the object `mysubtract_mdarr` with a two-dimension array from the object `my_fmdarr` with a two-dimension array and prints the values of elements to stdout for confirmation. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
mdarray mysubtract_mdarr(FLOAT_ZT, 2,2);

float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));
float mysubt_float[] = {100, 200, 300, 400};
mysubtract_mdarr.putdata((const void *)mysubt_float, sizeof(mysubt_float));

my_fmdarr.subtract(mysubtract_mdarr);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                   i, j, my_fmdarr.f(i, j));
    }
}
```

Output:

```
my_fmdarr value(0,0)...  [900.000000]
my_fmdarr value(1,0)...  [1800.000000]
my_fmdarr value(0,1)...  [2700.000000]
my_fmdarr value(1,1)...  [3600.000000]
```

4.5.56 multiply()

NAME

multiply() — Multiply an array object (for image data)

SYNOPSIS

```
mdarray &multiply( const mdarray &src_img, ssize_t dest_col = 0,
                   ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

This member function multiplies array elements of the object itself by those of the object `src_img`. A start position for multiplication can be specified separately for columns, rows, and layers. This member function is for image data.

PARAMETER

[I]	`src_img`	An instance of the class "mdarray" by which a subarray of this object is multiplied
[I]	`dest_col`	Subscript specifying the first column of the subarray
[I]	`dest_row`	Subscript specifying the first row of the subarray
[I]	`dest_layer`	Subscript specifying the first layer of the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code multiplies the object `my_fmdarr` with a two-dimension array by the object `mymulti_fmdarr` with a two-dimension array and prints the values of elements to stdout for confirmation:

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
float my_float[] = {1, 2, 3, 4};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

mdarray mymulti_fmdarr(FLOAT_ZT, 2,2);
float mymulti_float[] = {10, 20, 30, 40};
mymulti_fmdarr.putdata((const void *)mymulti_float, sizeof(mymulti_float));

my_fmdarr.multiply(mymulti_fmdarr);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                   i, j, my_fmdarr.f(i, j));
    }
}
```

Output:

```
my_fmdarr value(0,0)...  [10.000000]
my_fmdarr value(1,0)...  [40.000000]
my_fmdarr value(0,1)...  [90.000000]
my_fmdarr value(1,1)...  [160.000000]
```

4.5.57 divide()

NAME

divide() — Divide an array object (for image data)

SYNOPSIS

```
mdarray &divide( const mdarray &src_img, ssize_t dest_col = 0,
                 ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

This member function divides array elements of the object itself by those of the object `src_img`. A start position for division can be specified separately for columns, rows, and layers. This member function is for image data.

PARAMETER

[I]	`src_img`	An instance of the class "mdarray" by which a subarray of this object is divided
[I]	`dest_col`	Subscript for the first column of the subarray
[I]	`dest_row`	Subscript for the first row of the subarray
[I]	`dest_layer`	Subscript for the first layer of the subarray

([I] : input, [O] : output)

RETURN VALUE

A reference to itself

EXAMPLE

The following code divides the object `my_fmdarr` with a two-dimension array by the object `mydiv_mdarrf` with a two-dimension array and prints the values of elements to stdout for confirmation. For more information about `putdata()`, see §4.5.22.

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

mdarray mydiv_mdarrf(FLOAT_ZT, 2,2);
float mydiv_float[] = {2, 4, 6, 8};
mydiv_mdarrf.putdata((const void *)mydiv_float, sizeof(mydiv_float));

my_fmdarr.divide(mydiv_mdarrf);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
        i, j, my_fmdarr.f(i, j));
    }
}
```

Output:

```
my_fmdarr value(0,0)...  [500.000000]
my_fmdarr value(1,0)...  [500.000000]
my_fmdarr value(0,1)...  [500.000000]
my_fmdarr value(1,1)...  [500.000000]
```