

bml

2.2.0

Generated by Doxygen 1.8.17



<b>1 Basic Matrix Library (bml)</b>	<b>1</b>
1.1 Usage Examples	1
1.2 Modifying the library itself	1
1.3 Planned Features	1
<b>2 Future Plans</b>	<b>3</b>
2.1 Matrix Types	3
2.2 Precisions	3
2.3 Functions	3
<b>3 C Usage</b>	<b>5</b>
<b>4 Fortran Usage</b>	<b>7</b>
<b>5 Developer Documentation</b>	<b>9</b>
5.1 Developer Suggested Workflow	9
5.2 Coding Style	9
<b>6 FORTRAN TESTS</b>	<b>11</b>
6.1 Conventions and rules	11
<b>7 C TEST</b>	<b>13</b>
7.1 Compiling, running and checking the test	15
7.2 ADDING A FORTRAN TEST	15
<b>8 Module Index</b>	<b>17</b>
8.1 Modules	17
<b>9 Class Index</b>	<b>19</b>
9.1 Class List	19
<b>10 File Index</b>	<b>21</b>
10.1 File List	21
<b>11 Module Documentation</b>	<b>23</b>
11.1 Allocation and Deallocation Functions (C interface)	23
11.1.1 Detailed Description	23
11.1.2 Function Documentation	23
11.1.2.1 bml_allocate_memory()	23
11.1.2.2 bml_allocated()	24
11.1.2.3 bml_banded_matrix()	25
11.1.2.4 bml_clear()	25
11.1.2.5 bml_deallocate()	26
11.1.2.6 bml_deallocate_domain()	26
11.1.2.7 bml_default_domain()	27
11.1.2.8 bml_free_memory()	27

11.1.2.9 bml_free_ptr()	28
11.1.2.10 bml_identity_matrix()	28
11.1.2.11 bml_noinit_allocate_memory()	29
11.1.2.12 bml_noinit_matrix()	29
11.1.2.13 bml_noinit_rectangular_matrix()	30
11.1.2.14 bml_random_matrix()	31
11.1.2.15 bml_reallocate_memory()	31
11.1.2.16 bml_update_domain()	32
11.1.2.17 bml_zero_matrix()	32
11.2 Add Functions (C interface)	35
11.2.1 Detailed Description	35
11.2.2 Function Documentation	35
11.2.2.1 bml_add()	35
11.2.2.2 bml_add_identity()	36
11.2.2.3 bml_add_norm()	36
11.2.2.4 bml_scale_add_identity()	37
11.3 Converting between Matrix Formats (C interface)	38
11.3.1 Detailed Description	38
11.3.2 Function Documentation	38
11.3.2.1 bml_export_to_dense()	38
11.3.2.2 bml_import_from_dense()	39
11.4 Allocation and Deallocation Functions (Fortran interface)	40
11.5 Add Functions (Fortran interface)	41
11.6 Converting between Matrix Formats (Fortran interface)	42
<b>12 Class Documentation</b>	<b>43</b>
12.1 bml_domain_t Struct Reference	43
12.1.1 Detailed Description	43
12.1.2 Member Data Documentation	43
12.1.2.1 globalRowExtent	43
12.1.2.2 globalRowMax	44
12.1.2.3 globalRowMin	44
12.1.2.4 localDispl	44
12.1.2.5 localElements	44
12.1.2.6 localRowExtent	44
12.1.2.7 localRowMax	44
12.1.2.8 localRowMin	44
12.1.2.9 maxLocalExtent	44
12.1.2.10 minLocalExtent	45
12.1.2.11 totalCols	45
12.1.2.12 totalProcs	45
12.1.2.13 totalRows	45

12.2 bml_matrix_dimension_t Struct Reference	45
12.2.1 Detailed Description	45
12.2.2 Member Data Documentation	46
12.2.2.1 bsizes	46
12.2.2.2 N_cols	46
12.2.2.3 N_nz_max	46
12.2.2.4 N_rows	46
12.2.2.5 NB	46
<b>13 File Documentation</b>	<b>47</b>
13.1 /tmp/bml/src/C-interface/bml.h File Reference	47
13.1.1 Detailed Description	47
13.2 /tmp/bml/src/C-interface/bml_add.h File Reference	48
13.3 /tmp/bml/src/C-interface/bml_adjungate_triangle.h File Reference	48
13.3.1 Function Documentation	49
13.3.1.1 bml_adjungate_triangle()	49
13.4 /tmp/bml/src/C-interface/bml_allocate.h File Reference	50
13.5 /tmp/bml/src/C-interface/bml_convert.h File Reference	51
13.5.1 Function Documentation	52
13.5.1.1 bml_convert()	52
13.6 /tmp/bml/src/C-interface/bml_copy.h File Reference	52
13.6.1 Function Documentation	53
13.6.1.1 bml_copy()	53
13.6.1.2 bml_copy_domain()	54
13.6.1.3 bml_copy_new()	54
13.6.1.4 bml_reorder()	55
13.6.1.5 bml_restore_domain()	56
13.6.1.6 bml_save_domain()	56
13.7 /tmp/bml/src/C-interface/bml_element_multiply.h File Reference	56
13.7.1 Function Documentation	57
13.7.1.1 bml_element_multiply_AB()	57
13.8 /tmp/bml/src/C-interface/bml_export.h File Reference	58
13.9 /tmp/bml/src/C-interface/bml_getters.h File Reference	59
13.9.1 Function Documentation	60
13.9.1.1 bml_get_diagonal()	60
13.9.1.2 bml_get_element()	61
13.9.1.3 bml_get_row()	62
13.10 /tmp/bml/src/C-interface/bml_import.h File Reference	62
13.11 /tmp/bml/src/C-interface/bml_init.h File Reference	63
13.11.1 Function Documentation	64
13.11.1.1 bml_init()	64
13.11.1.2 bml_initF()	64

13.12 /tmp/bml/src/C-interface/bml_introspection.h File Reference . . . . .	65
13.12.1 Function Documentation . . . . .	66
13.12.1.1 bml_get_bandwidth() . . . . .	66
13.12.1.2 bml_get_deep_type() . . . . .	66
13.12.1.3 bml_get_distribution_mode() . . . . .	67
13.12.1.4 bml_get_M() . . . . .	68
13.12.1.5 bml_get_N() . . . . .	68
13.12.1.6 bml_get_precision() . . . . .	69
13.12.1.7 bml_get_row_bandwidth() . . . . .	70
13.12.1.8 bml_get_sparsity() . . . . .	71
13.12.1.9 bml_get_type() . . . . .	72
13.13 /tmp/bml/src/C-interface/bml_logger.h File Reference . . . . .	72
13.13.1 Macro Definition Documentation . . . . .	73
13.13.1.1 LOG_DEBUG . . . . .	73
13.13.1.2 LOG_ERROR . . . . .	74
13.13.1.3 LOG_INFO . . . . .	74
13.13.1.4 LOG_WARN . . . . .	74
13.13.2 Enumeration Type Documentation . . . . .	74
13.13.2.1 bml_log_level_t . . . . .	74
13.13.3 Function Documentation . . . . .	74
13.13.3.1 bml_log() . . . . .	75
13.13.3.2 bml_log_location() . . . . .	75
13.14 /tmp/bml/src/C-interface/bml_multiply.h File Reference . . . . .	75
13.14.1 Function Documentation . . . . .	76
13.14.1.1 bml_multiply() . . . . .	77
13.14.1.2 bml_multiply_AB() . . . . .	77
13.14.1.3 bml_multiply_adjust_AB() . . . . .	78
13.14.1.4 bml_multiply_x2() . . . . .	78
13.15 /tmp/bml/src/C-interface/bml_norm.h File Reference . . . . .	79
13.15.1 Function Documentation . . . . .	80
13.15.1.1 bml_fnorm() . . . . .	80
13.15.1.2 bml_fnorm2() . . . . .	81
13.15.1.3 bml_sum_AB() . . . . .	82
13.15.1.4 bml_sum_squares() . . . . .	82
13.15.1.5 bml_sum_squares2() . . . . .	83
13.15.1.6 bml_sum_squares_submatrix() . . . . .	84
13.16 /tmp/bml/src/C-interface/bml_normalize.h File Reference . . . . .	84
13.16.1 Function Documentation . . . . .	85
13.16.1.1 bml_gershgorin() . . . . .	85
13.16.1.2 bml_gershgorin_partial() . . . . .	86
13.16.1.3 bml_normalize() . . . . .	86
13.17 /tmp/bml/src/C-interface/bml_parallel.h File Reference . . . . .	87

13.17.1 Function Documentation	88
13.17.1.1 bml_allGatherVParallel()	88
13.17.1.2 bml_getMyRank()	88
13.17.1.3 bml_getNRanks()	89
13.18 /tmp/bml/src/C-interface/bml_scale.h File Reference	89
13.18.1 Function Documentation	90
13.18.1.1 bml_scale()	90
13.18.1.2 bml_scale_inplace()	91
13.18.1.3 bml_scale_new()	91
13.19 /tmp/bml/src/C-interface/bml_setters.h File Reference	92
13.20 /tmp/bml/src/C-interface/bml_shutdown.h File Reference	93
13.20.1 Function Documentation	94
13.20.1.1 bml_shutdown()	94
13.20.1.2 bml_shutdownF()	94
13.21 /tmp/bml/src/C-interface/bml_submatrix.h File Reference	95
13.21.1 Function Documentation	96
13.21.1.1 bml_adjacency()	96
13.21.1.2 bml_adjacency_group()	96
13.21.1.3 bml_group_matrix()	97
13.21.1.4 bml_matrix2submatrix()	98
13.21.1.5 bml_matrix2submatrix_index()	98
13.21.1.6 bml_matrix2submatrix_index_graph()	99
13.21.1.7 bml_submatrix2matrix()	100
13.22 /tmp/bml/src/C-interface/bml_threshold.h File Reference	100
13.22.1 Function Documentation	101
13.22.1.1 bml_threshold()	101
13.22.1.2 bml_threshold_new()	102
13.23 /tmp/bml/src/C-interface/bml_trace.h File Reference	102
13.23.1 Function Documentation	103
13.23.1.1 bml_trace()	103
13.23.1.2 bml_trace_mult()	104
13.24 /tmp/bml/src/C-interface/bml_transpose.h File Reference	105
13.24.1 Function Documentation	105
13.24.1.1 bml_transpose()	105
13.24.1.2 bml_transpose_new()	106
13.25 /tmp/bml/src/C-interface/bml_transpose_triangle.h File Reference	107
13.25.1 Function Documentation	107
13.25.1.1 bml_transpose_triangle()	107
13.26 /tmp/bml/src/C-interface/bml_types.h File Reference	108
13.26.1 Typedef Documentation	108
13.26.1.1 bml_matrix_t	109
13.26.1.2 bml_vector_t	109

---

13.26.2 Enumeration Type Documentation . . . . .	109
13.26.2.1 bml_dense_order_t . . . . .	109
13.26.2.2 bml_distribution_mode_t . . . . .	109
13.26.2.3 bml_matrix_precision_t . . . . .	109
13.26.2.4 bml_matrix_type_t . . . . .	110
13.27 /tmp/bml/src/C-interface/bml_types_private.h File Reference . . . . .	110
13.28 /tmp/bml/src/C-interface/bml_utilities.h File Reference . . . . .	110
13.28.1 Function Documentation . . . . .	111
13.28.1.1 bml_print_bml_matrix() . . . . .	111
13.28.1.2 bml_print_bml_vector() . . . . .	112
13.28.1.3 bml_print_dense_matrix() . . . . .	113
13.28.1.4 bml_print_dense_vector() . . . . .	114
13.28.1.5 bml_read_bml_matrix() . . . . .	114
13.28.1.6 bml_write_bml_matrix() . . . . .	114
<b>Index</b>	<b>117</b>



# Chapter 1

## Basic Matrix Library (bml)

This library implements a common API for linear algebra and matrix functions in C and Fortran. It offers several data structures for matrix storage and algorithms. Currently the following matrix data types are implemented:

- dense
- ellpack (sparse)
- csr (sparse)
- ellblock (sparse)
- ellsort (sparse)

### 1.1 Usage Examples

Usage examples can be found here:

- [Fortran Usage](#)
- [C Usage](#)

### 1.2 Modifying the library itself

If you are interested in modifying the library code itself, please have a look at the [Developer Documentation](#).

### 1.3 Planned Features

We are planning to eventually support different matrix types and matrix operations on a variety of hardware platforms. For details, please have a look at our [future plans](#).

Copyright

Los Alamos National Laboratory 2015



## Chapter 2

# Future Plans

### 2.1 Matrix Types

Support types:

- `bml_matrix_t`
- Colinear
- Noncolinear

### 2.2 Precisions

The bml supports the following precisions:

- logical (for matrix masks)
- single real
- double real
- single complex
- double complex

### 2.3 Functions

The library supports the following matrix operations:

- Format Conversion
  - `bml_import::bml_import_from_dense`
  - `bml_export::bml_export_to_dense`
  - `bml_convert::bml_convert`
- Masking

- Masked operations (restricted to a subgraph)
- Addition
  - $\alpha A + \beta B$ : `bml_add::bml_add`
  - $\alpha A + \beta$ : `bml_add::bml_add_identity`
- Copy
  - $B \leftarrow A$ : `bml_copy::bml_copy`
- Diagonalize
  - `bml_diagonalize::bml_diagonalize`
- Introspection
  - `bml_introspection::bml_get_type`
  - `bml_introspection::bml_get_size`
  - `bml_introspection::bml_get_bandwidth`
  - `bml_introspection::bml_get_spectral_range`
  - `bml_introspection::bml_get_HOMO_LUMO`
- Matrix manipulation:
  - `bml_get::bml_get`
  - `bml_get::bml_get_rows`
  - `bml_set::bml_set`
  - `bml_set::bml_set_rows`
- Multiplication
  - $\alpha A \times B + \beta C$ : `bml_multiply::bml_multiply`
- Printing
  - `bml_utilities::bml_print_matrix`
- Scaling
  - $A \leftarrow \alpha A$ : `bml_scale::bml_scale_one`
  - $B \leftarrow \alpha A$ : `bml_scale::bml_scale_two`
- Matrix trace
  - $\text{Tr}[A]$ : `bml_trace::bml_trace`
  - $\text{Tr}[AB]$ : `bml_trace::bml_product_trace`
- Matrix norm
  - 2-norm
  - Frobenius norm
- Matrix transpose
  - `bml_transpose::bml_transpose`
- Matrix commutator/anticommutator
  - `bml_commutator::bml_commutator`
  - `bml_commutator::bml_anticommutator`

Back to the [main page](#).

## Chapter 3

# C Usage

In C, the following example code does the same as the above Fortran code:

```
#include <bml.h>
bml_matrix_t *A = bml_zero_matrix(dense, single_real, 100);
bml_deallocate(&A);
```

Back to the [main page](#).



## Chapter 4

# Fortran Usage

The use of this library is pretty straightforward. In the application code, use the bml main module,

```
use bml
```

A matrix is of type

```
type(bml_matrix_t) :: a
```

There are two important things to note. First, although not explicitly state in the above example, the matrix is not yet allocated. Hence, the matrix needs to be allocated through an allocation procedure with the desired type and precision, e.g. dense:double, see the page on [allocation functions](#) for a complete list. For instance,

```
call bml_zero_matrix(BML_MATRIX_DENSE, BML_PRECISION_DOUBLE, 100, a)
```

will allocate a dense, double-precision,  $100 \times 100$  matrix which is initialized to zero. Additional functions allocate special matrices,

- `bml_allocate::bml_random_matrix` Allocate and initialize a random matrix.
- `bml_allocate::bml_identity_matrix` Allocate and initialize the identity matrix.

A matrix is deallocated by calling

```
call bml_deallocate(a)
```

Back to the [main page](#).





## Chapter 5

# Developer Documentation

### 5.1 Developer Suggested Workflow

We try to preserve a linear history in our main (master) branch. Instead of pulling (i.e. merging), we suggest you use:

```
$ git pull --rebase
```

And then

```
$ git push
```

To push your changes back to the server.

### 5.2 Coding Style

Please indent your C code using

```
$ indent -gnu -nut -i4 -bli0
```

Back to the [main page](#).



## Chapter 6

# FORTRAN TESTS

The tests are driven by a general executable created when the code is compiled with `BML_TESTING=yes`. This driver is called `bml-testf` compiled with the `testf.F90` source.

Every low level source code of the type `name_typed.F90` is pre-processed using the `/scripts/convert-template.in` to change to the particular element kind and precision. Two dummy variables are used:

- `DUMMY_KIND`: That gets replaced with either `real` or `complex`
- `DUMMY_PREC` or `_MP`: That gets replaced with `SP/_SP` of `DP/_DP` (defined in `prec.F90`)

There are `example_template*` files that can be used as starting point to add a particular test.

### 6.1 Conventions and rules

The general driver takes four variables (this can be extended as needed). These variables are:

- `test_name`: The name of the test
- `matrix_type`: The matrix format (matrix format and matrix type are the same thing)
- `element_type`: The element "kind" and "precision". For example `double_real`, which gets converted to `real(8)` at the lowest level.

NOTE: Try to be as explicit as possible in naming the variables.



## Chapter 7

# C TEST

It is essential to add a proper test for each function we create. We would even recommend to add a test before adding the functionality to have a piece of code that could be executed. To do this, we have provided this step-by-step tutorial. Let's consider that we are adding a test which name is "mytest".

We will first modify the three following files accordingly by adding the name of the test in them. Note: Whenever we can we will proceed to add names/files in alphabetical order to keep consistency in the source file.

The three files that need to be modified are:

- /tests/CMakeLists.txt
- /tests/bml\_test.c
- /tests/bml\_test.h

In CMakeLists.txt we will add the test name in three places:

```
set (SOURCES_Typed
    test1_typed.c
    ...
    mytest_typed.c
    ...
    testN_typed.c)

;

add_executable (bml-test
    test1.c
    ...
    mytest.c
    ...
    testN.c)
```

and

```
foreach(N add test1 ... mytest ... testN)
```

Second, we should modify the bml\_test.h to include our "future" header file. We will add the name as follows:

```
#include "test1.h"
...
#include "mytest.h"
...
#include "testN.h"
```

Finally, we will modify the `bml_test.c` file in four positions. We will first indicate that there is going to be an extra test by increasing the `NUM_TEST` variable:

```
const int NUM_TESTS = <N>;
```

where `N` has to be replaced by the total number of tests. Next we will add the test name in the `test_name` array:

```
const char *test_name[] =
    { "test1", ... , "mytest", ... , "testN" }
```

Please ensure that the number of entries in `test_name`, `test_description`, and `testers` matches the value of `NUM_TESTS`. This will be followed by a description of the test:

```
const char *test_description[] = {
    "Description of test 1",
    ....
    "Description of mytest",
    ....
    "Description of test N" }
```

And finally we will add the name of the function that will perform the test:

```
const test_function_t testers[] = {
    test_test1,
    ...
    test_mytest,
    ...
    test_testN }
```

After this is done we will start creating the source code for our test. These files will be created inside `/tests/` and will be named as follows:

- `/tests/mytest.c`
- `/tests/mytest.h`
- `/tests/mytest_typed.c`

This means that for each test we will have a "header file" (`mytest.h`), a "driver" (`mytest.c`) and a typed (`mytest_typed.c`). In this last file we will add all the functionalities for testing (actual test). For these three files we provide templates which names are `template.c`, `template.h` and `template_typed.c`. These files (template-) will have to be renamed to (`mytest-`). The final step which is left to the developer is to add some lines of code inside `mytest_typed.c` to make the test work. For example, this can be a difference between two values that has to be less than a tolerance.

## 7.1 Compiling, running and checking the test

Once the functionality is added we need to make sure that the test is compiling, running and passing. For this we can do the following:

First we can try to configure the code using the example `build.sh` file located inside the main directory. Second, if the configuration proceeds with no error we build the code:

```
$ ./example_build
$ cd build; make
```

If everything is built without problems. We can test the whole code:

```
$ make test
```

or if we want to see details of the test:

```
$ make test ARGS="-V"
```

We can check if the new test we have added appears in the list of tests.

If we want to run just the test we have created we can do:

```
$ cd /build/tests
$ ./bml-test -n mytest -t ellpack -p double_complex
```

The latter means that we will run our test with `ellpack` matrix type and `double_complex` precision. Once the test passes for every precision and matrix type we will need to make sure there are no memory leaks in the test or routine. For this we could run `valgrind` as following:

```
$ valgrind ./bml-test -n mytest -t ellpack -p double_complex
```

You can also trigger tests by running `ctest` directly.

```
$ cd build $ ctest -R mytest --output-on-failure
```

After all the tests passed, we should indent the new files using the `indent.sh` Running `indent.sh` (located in the main folder) will indent all files.

```
$ ./indent.sh
```

## 7.2 ADDING A FORTRAN TEST





## Chapter 8

# Module Index

### 8.1 Modules

Here is a list of all modules:

Allocation and Deallocation Functions (C interface) . . . . .	<a href="#">23</a>
Add Functions (C interface) . . . . .	<a href="#">35</a>
Converting between Matrix Formats (C interface) . . . . .	<a href="#">38</a>
Allocation and Deallocation Functions (Fortran interface) . . . . .	<a href="#">40</a>
Add Functions (Fortran interface) . . . . .	<a href="#">41</a>
Converting between Matrix Formats (Fortran interface) . . . . .	<a href="#">42</a>



## Chapter 9

# Class Index

### 9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">bml_domain_t</a> . . . . .	43
<a href="#">bml_matrix_dimension_t</a> . . . . .	45



## Chapter 10

# File Index

### 10.1 File List

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

/tmp/bml/src/C-interface/ <b>blas.h</b>	??
/tmp/bml/src/C-interface/ <b>bml.h</b>	47
/tmp/bml/src/C-interface/ <b>bml_add.h</b>	48
/tmp/bml/src/C-interface/ <b>bml_adjungate_triangle.h</b>	48
/tmp/bml/src/C-interface/ <b>bml_allocate.h</b>	50
/tmp/bml/src/C-interface/ <b>bml_convert.h</b>	51
/tmp/bml/src/C-interface/ <b>bml_copy.h</b>	52
/tmp/bml/src/C-interface/ <b>bml_diagonalize.h</b>	??
/tmp/bml/src/C-interface/ <b>bml_element_multiply.h</b>	56
/tmp/bml/src/C-interface/ <b>bml_elemental.h</b>	??
/tmp/bml/src/C-interface/ <b>bml_export.h</b>	58
/tmp/bml/src/C-interface/ <b>bml_getters.h</b>	59
/tmp/bml/src/C-interface/ <b>bml_import.h</b>	62
/tmp/bml/src/C-interface/ <b>bml_init.h</b>	63
/tmp/bml/src/C-interface/ <b>bml_introspection.h</b>	65
/tmp/bml/src/C-interface/ <b>bml_inverse.h</b>	??
/tmp/bml/src/C-interface/ <b>bml_logger.h</b>	72
/tmp/bml/src/C-interface/ <b>bml_multiply.h</b>	75
/tmp/bml/src/C-interface/ <b>bml_norm.h</b>	79
/tmp/bml/src/C-interface/ <b>bml_normalize.h</b>	84
/tmp/bml/src/C-interface/ <b>bml_parallel.h</b>	87
/tmp/bml/src/C-interface/ <b>bml_scale.h</b>	89
/tmp/bml/src/C-interface/ <b>bml_setters.h</b>	92
/tmp/bml/src/C-interface/ <b>bml_shutdown.h</b>	93
/tmp/bml/src/C-interface/ <b>bml_submatrix.h</b>	95
/tmp/bml/src/C-interface/ <b>bml_threshold.h</b>	100
/tmp/bml/src/C-interface/ <b>bml_trace.h</b>	102
/tmp/bml/src/C-interface/ <b>bml_transpose.h</b>	105
/tmp/bml/src/C-interface/ <b>bml_transpose_triangle.h</b>	107
/tmp/bml/src/C-interface/ <b>bml_types.h</b>	108
/tmp/bml/src/C-interface/ <b>bml_types_private.h</b>	110
/tmp/bml/src/C-interface/ <b>bml_utilities.h</b>	110
/tmp/bml/src/C-interface/ <b>lapack.h</b>	??



# Chapter 11

## Module Documentation

### 11.1 Allocation and Deallocation Functions (C interface)

#### Functions

- `int bml_allocated (bml_matrix_t *A)`
- `void * bml_allocate_memory (size_t size)`
- `void * bml_noinit_allocate_memory (size_t size)`
- `void * bml_reallocate_memory (void *ptr, const size_t size)`
- `void bml_free_memory (void *ptr)`
- `void bml_free_ptr (void **ptr)`
- `void bml_deallocate (bml_matrix_t **A)`
- `void bml_deallocate_domain (bml_domain_t *D)`
- `void bml_clear (bml_matrix_t *A)`
- `bml_matrix_t * bml_noinit_rectangular_matrix (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, bml_matrix_dimension_t matrix_dimension, bml_distribution_mode_t distrib_mode)`
- `bml_matrix_t * bml_noinit_matrix (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, int N, int M, bml_distribution_mode_t distrib_mode)`
- `bml_matrix_t * bml_zero_matrix (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, int N, int M, bml_distribution_mode_t distrib_mode)`
- `bml_matrix_t * bml_random_matrix (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, int N, int M, bml_distribution_mode_t distrib_mode)`
- `bml_matrix_t * bml_banded_matrix (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, int N, int M, bml_distribution_mode_t distrib_mode)`
- `bml_matrix_t * bml_identity_matrix (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, int N, int M, bml_distribution_mode_t distrib_mode)`
- `bml_domain_t * bml_default_domain (int N, int M, bml_distribution_mode_t distrib_mode)`
- `void bml_update_domain (bml_matrix_t *A, int *localPartMin, int *localPartMax, int *nnodesInPart)`

#### 11.1.1 Detailed Description

#### 11.1.2 Function Documentation

##### 11.1.2.1 `bml_allocate_memory()`

```
void* bml_allocate_memory (
    size_t size )
```

Allocate and zero a chunk of memory.

**Parameters**

<i>size</i>	The size of the memory.
-------------	-------------------------

**Returns**

A pointer to the allocated chunk.

Here is the caller graph for this function:

**11.1.2.2 bml\_allocated()**

```
int bml_allocated (
    bml_matrix_t * A )
```

Check if matrix is allocated.

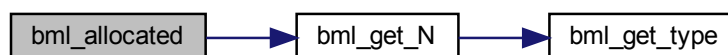
**Parameters**

<i>A[in,out]</i>	Matrix
------------------	--------

**Returns**

> 0 if allocated, else -1

Here is the call graph for this function:





### 11.1.2.3 bml\_banded\_matrix()

```
bml_matrix_t* bml_banded_matrix (
    bml_matrix_type_t matrix_type,
    bml_matrix_precision_t matrix_precision,
    int N,
    int M,
    bml_distribution_mode_t distrib_mode )
```

Allocate a banded matrix.

Note that the matrix  $A$  will be newly allocated. The function does not check whether the matrix is already allocated.

#### Parameters

<i>matrix_type</i>	The matrix type.
<i>matrix_precision</i>	The precision of the matrix.
<i>N</i>	The matrix size.
<i>M</i>	The bandwidth of the matrix.
<i>distrib_mode</i>	The distribution mode.

#### Returns

The matrix.

### 11.1.2.4 bml\_clear()

```
void bml_clear (
    bml_matrix_t * A )
```

Clear a matrix.

#### Parameters

<i>A[in,out]</i>	The matrix.
------------------	-------------

Here is the call graph for this function:



### 11.1.2.5 bml\_deallocate()

```
void bml_deallocate (
    bml_matrix_t ** A )
```

Deallocate a matrix.

#### Parameters

<i>A[in,out]</i>	The matrix.
------------------	-------------

Here is the call graph for this function:



### 11.1.2.6 bml\_deallocate\_domain()

```
void bml_deallocate_domain (
    bml_domain_t * D )
```

Deallocate a domain.

#### Parameters

<i>D[in,out]</i>	The domain.
------------------	-------------

Here is the call graph for this function:



### 11.1.2.7 bml\_default\_domain()

```
bml_domain_t* bml_default_domain (
    int N,
    int M,
    bml_distribution_mode_t distrib_mode )
```

Allocate a default domain for a bml matrix.

#### Parameters

<i>N</i>	The number of rows
<i>M</i>	The number of columns
<i>distrib_mode</i>	The distribution mode

#### Returns

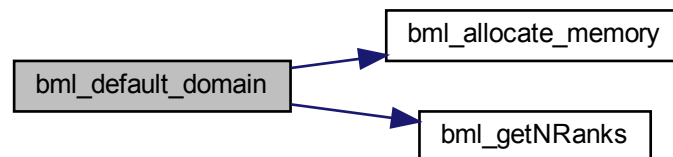
The domain

For first rank

For middle ranks

For last rank

Number of elements and displacement per rankHere is the call graph for this function:



### 11.1.2.8 bml\_free\_memory()

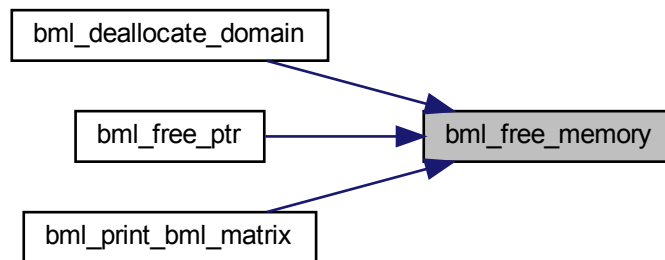
```
void bml_free_memory (
    void * ptr )
```

Deallocate a chunk of memory.

#### Parameters

<i>ptr</i>	A pointer to the previously allocated chunk.
------------	--

Here is the caller graph for this function:



#### 11.1.2.9 bml\_free\_ptr()

```
void bml_free_ptr (
    void ** ptr )
```

De-allocate a chunk of memory that was allocated inside a C function. This is used by the Fortran `bml_free_C` interface. Note the "pointer to pointer" in the API.

##### Parameters

<i>ptr</i>	A pointer to the previously allocated chunk.
------------	--

Here is the call graph for this function:



#### 11.1.2.10 bml\_identity\_matrix()

```
bml_matrix_t* bml_identity_matrix (
    bml_matrix_type_t matrix_type,
```

```

bml_matrix_precision_t matrix_precision,
int N,
int M,
bml_distribution_mode_t distrib_mode )

```

Allocate the identity matrix.

Note that the matrix  $A$  will be newly allocated. The function does not check whether the matrix is already allocated.

#### Parameters

<i>matrix_type</i>	The matrix type.
<i>matrix_precision</i>	The precision of the matrix.
<i>N</i>	The matrix size.
<i>M</i>	The number of non-zeroes per row.
<i>distrib_mode</i>	The distribution mode.

#### Returns

The matrix.

#### 11.1.2.11 bml\_noinit\_allocate\_memory()

```

void* bml_noinit_allocate_memory (
    size_t size )

```

Allocate a chunk of memory without initialization.

#### Parameters

<i>size</i>	The size of the memory.
-------------	-------------------------

#### Returns

A pointer to the allocated chunk.

#### 11.1.2.12 bml\_noinit\_matrix()

```

bml_matrix_t* bml_noinit_matrix (
    bml_matrix_type_t matrix_type,
    bml_matrix_precision_t matrix_precision,
    int N,
    int M,
    bml_distribution_mode_t distrib_mode )

```

Allocate a matrix without initializing.

Note that the matrix  $A$  will be newly allocated. The function does not check whether the matrix is already allocated.

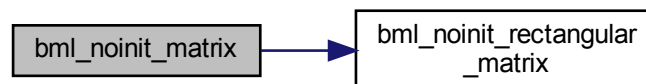
## Parameters

<i>matrix_type</i>	The matrix type.
<i>matrix_precision</i>	The precision of the matrix.
<i>N</i>	The matrix size.
<i>M</i>	The number of non-zeroes per row.
<i>distrib_mode</i>	The distribution mode.

## Returns

The matrix.

Here is the call graph for this function:



## 11.1.2.13 bml\_noinit\_rectangular\_matrix()

```

bml_matrix_t* bml_noinit_rectangular_matrix (
    bml_matrix_type_t matrix_type,
    bml_matrix_precision_t matrix_precision,
    bml_matrix_dimension_t matrix_dimension,
    bml_distribution_mode_t distrib_mode )
  
```

Allocate a matrix without initializing.

Note that the matrix  $A$  will be newly allocated. The function does not check whether the matrix is already allocated.

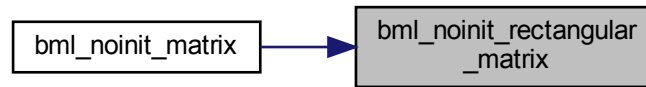
## Parameters

<i>matrix_type</i>	The matrix type.
<i>matrix_precision</i>	The precision of the matrix.
<i>matrix_dimension</i>	The matrix size.
<i>distrib_mode</i>	The distribution mode.

## Returns

The matrix.

Here is the caller graph for this function:



#### 11.1.2.14 bml\_random\_matrix()

```

bml_matrix_t* bml_random_matrix (
    bml_matrix_type_t matrix_type,
    bml_matrix_precision_t matrix_precision,
    int N,
    int M,
    bml_distribution_mode_t distrib_mode )
  
```

Allocate a random matrix.

Note that the matrix  $A$  will be newly allocated. The function does not check whether the matrix is already allocated.

##### Parameters

<i>matrix_type</i>	The matrix type.
<i>matrix_precision</i>	The precision of the matrix.
<i>N</i>	The matrix size.
<i>M</i>	The number of non-zeroes per row.
<i>distrib_mode</i>	The distribution mode.

##### Returns

The matrix.

#### 11.1.2.15 bml\_reallocate\_memory()

```

void* bml_reallocate_memory (
    void * ptr,
    const size_t size )
  
```

Reallocate a chunk of memory.

## Parameters

<i>size</i>	The size of the memory.
-------------	-------------------------

## Returns

A pointer to the reallocated chunk.

11.1.2.16 **bml\_update\_domain()**

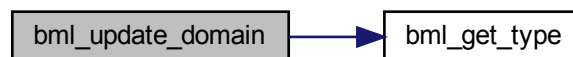
```
void bml_update_domain (
    bml_matrix_t * A,
    int * localPartMin,
    int * localPartMax,
    int * nnodesInPart )
```

Update a domain for a bml matrix.

## Parameters

<i>A</i>	Matrix with domain
<i>localPartMin</i>	First part on each rank
<i>localPartMax</i>	Last part on each rank
<i>nnodesInPart</i>	Number of nodes in each part

Here is the call graph for this function:

11.1.2.17 **bml\_zero\_matrix()**

```
bml_matrix_t* bml_zero_matrix (
    bml_matrix_type_t matrix_type,
    bml_matrix_precision_t matrix_precision,
    int N,
    int M,
    bml_distribution_mode_t distrib_mode )
```



Allocate the zero matrix.

Note that the matrix  $A$  will be newly allocated. The function does not check whether the matrix is already allocated.

**Parameters**

<i>matrix_type</i>	The matrix type.
<i>matrix_precision</i>	The precision of the matrix.
<i>N</i>	The matrix size.
<i>M</i>	The number of non-zeroes per row.
<i>distrib_mode</i>	The distribution mode.

**Returns**

The matrix.

## 11.2 Add Functions (C interface)

### Functions

- void `bml_add` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, double alpha, double beta, double threshold)
- double `bml_add_norm` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, double alpha, double beta, double threshold)
- void `bml_add_identity` (`bml_matrix_t` \*A, double beta, double threshold)
- void `bml_scale_add_identity` (`bml_matrix_t` \*A, double alpha, double beta, double threshold)

### 11.2.1 Detailed Description

### 11.2.2 Function Documentation

#### 11.2.2.1 `bml_add()`

```
void bml_add (
    bml_matrix_t * A,
    bml_matrix_t * B,
    double alpha,
    double beta,
    double threshold )
```

Matrix addition.

$$A \leftarrow \alpha A + \beta B$$

#### Parameters

<i>A</i>	Matrix A
<i>B</i>	Matrix B
<i>alpha</i>	Scalar factor multiplied by A
<i>beta</i>	Scalar factor multiplied by B
<i>threshold</i>	Threshold for matrix addition

Here is the call graph for this function:



### 11.2.2.2 bml\_add\_identity()

```
void bml_add_identity (
    bml_matrix_t * A,
    double beta,
    double threshold )
```

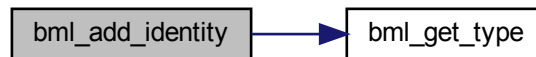
Matrix addition.

$$A \leftarrow A + \beta \text{Id}$$

#### Parameters

<i>A</i>	Matrix A
<i>beta</i>	Scalar factor multiplied by I
<i>threshold</i>	Threshold for matrix addition

Here is the call graph for this function:



### 11.2.2.3 bml\_add\_norm()

```
double bml_add_norm (
    bml_matrix_t * A,
    bml_matrix_t * B,
    double alpha,
    double beta,
    double threshold )
```

Matrix addition with calculation of TrNorm.

$$A \leftarrow \alpha A + \beta B$$

#### Parameters

<i>A</i>	Matrix A
<i>B</i>	Matrix B
<i>alpha</i>	Scalar factor multiplied by A
<i>beta</i>	Scalar factor multiplied by B
<i>threshold</i>	Threshold for matrix addition

Here is the call graph for this function:



#### 11.2.2.4 bml\_scale\_add\_identity()

```

void bml_scale_add_identity (
    bml_matrix_t * A,
    double alpha,
    double beta,
    double threshold )
  
```

Matrix addition.

$$A \leftarrow \alpha A + \beta \text{Id}$$

##### Parameters

<i>A</i>	Matrix A
<i>alpha</i>	Scalar factor multiplied by A
<i>beta</i>	Scalar factor multiplied by I
<i>threshold</i>	Threshold for matrix addition

Here is the call graph for this function:



## 11.3 Converting between Matrix Formats (C interface)

### Functions

- `void * bml_export_to_dense (bml_matrix_t *A, bml_dense_order_t order)`
- `bml_matrix_t * bml_import_from_dense (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, bml_dense_order_t order, int N, int M, void *A, double threshold, bml_distribution_mode_t distribution_mode)`

#### 11.3.1 Detailed Description

#### 11.3.2 Function Documentation

##### 11.3.2.1 bml\_export\_to\_dense()

```
void* bml_export_to_dense (
    bml_matrix_t * A,
    bml_dense_order_t order )
```

Export a bml matrix.

The returned pointer has to be typecase into the proper real type. If the bml matrix is a single precision matrix, then the following should be used:

```
float *A_dense = bml_export_to_dense(A_bml);
```

The matrix size can be queried with

```
int N = bml_get_size(A_bml);
```

#### Parameters

<i>A</i>	The bml matrix
<i>order</i>	The matrix element order

#### Returns

The dense matrix

Here is the call graph for this function:



Here is the caller graph for this function:



### 11.3.2.2 bml\_import\_from\_dense()

```

bml_matrix_t* bml_import_from_dense (
    bml_matrix_type_t matrix_type,
    bml_matrix_precision_t matrix_precision,
    bml_dense_order_t order,
    int N,
    int M,
    void * A,
    double threshold,
    bml_distribution_mode_t distrib_mode )
  
```

Import a dense matrix.

#### Parameters

<i>matrix_type</i>	The matrix type
<i>matrix_precision</i>	The real precision
<i>order</i>	The dense matrix element order
<i>N</i>	The number of rows/columns
<i>M</i>	The number of non-zeroes per row
<i>A</i>	The dense matrix
<i>threshold</i>	The matrix element magnited threshold

#### Returns

The bml matrix

## 11.4 Allocation and Deallocation Functions (Fortran interface)



## 11.5 Add Functions (Fortran interface)

## 11.6 Converting between Matrix Formats (Fortran interface)

## Chapter 12

# Class Documentation

### 12.1 bml\_domain\_t Struct Reference

```
#include <bml_types.h>
```

#### Public Attributes

- int [totalProcs](#)
- int [totalRows](#)
- int [totalCols](#)
- int [globalRowMin](#)
- int [globalRowMax](#)
- int [globalRowExtent](#)
- int [maxLocalExtent](#)
- int [minLocalExtent](#)
- int \* [localRowMin](#)
- int \* [localRowMax](#)
- int \* [localRowExtent](#)
- int \* [localElements](#)
- int \* [localDispl](#)

#### 12.1.1 Detailed Description

Decomposition for working in parallel.

#### 12.1.2 Member Data Documentation

##### 12.1.2.1 globalRowExtent

```
int bml_domain_t::globalRowExtent
```

global total rows

#### 12.1.2.2 globalRowMax

```
int bml_domain_t::globalRowMax
```

global maximum row number

#### 12.1.2.3 globalRowMin

```
int bml_domain_t::globalRowMin
```

global minimum row number

#### 12.1.2.4 localDispl

```
int* bml_domain_t::localDispl
```

local displacements per rank for 2D

#### 12.1.2.5 localElements

```
int* bml_domain_t::localElements
```

local number of elements per rank

#### 12.1.2.6 localRowExtent

```
int* bml_domain_t::localRowExtent
```

extent of rows per rank, localRowMax - localRowMin

#### 12.1.2.7 localRowMax

```
int* bml_domain_t::localRowMax
```

maximum row per rank

#### 12.1.2.8 localRowMin

```
int* bml_domain_t::localRowMin
```

minimum row per rank

#### 12.1.2.9 maxLocalExtent

```
int bml_domain_t::maxLocalExtent
```

maximum extent for most processors

### 12.1.2.10 minLocalExtent

```
int bml_domain_t::minLocalExtent
```

minimum extent for last processors

### 12.1.2.11 totalCols

```
int bml_domain_t::totalCols
```

total number of columns

### 12.1.2.12 totalProcs

```
int bml_domain_t::totalProcs
```

number of processors

### 12.1.2.13 totalRows

```
int bml_domain_t::totalRows
```

total number of rows

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

- /tmp/bml/src/C-interface/[bml\\_types.h](#)

## 12.2 bml\_matrix\_dimension\_t Struct Reference

```
#include <bml_types.h>
```

### Public Attributes

- int [N\\_rows](#)
- int [N\\_cols](#)
- int [N\\_nz\\_max](#)
- int \* [bsizes](#)
- int [NB](#)

### 12.2.1 Detailed Description

The matrix dimensions.

## 12.2.2 Member Data Documentation

### 12.2.2.1 bsizes

```
int* bml_matrix_dimension_t::bsizes
```

The block sizes (for block\_ellpack).

### 12.2.2.2 N\_cols

```
int bml_matrix_dimension_t::N_cols
```

The number of columns.

### 12.2.2.3 N\_nz\_max

```
int bml_matrix_dimension_t::N_nz_max
```

The maximum number of non-zeros per row (for ellpack).

### 12.2.2.4 N\_rows

```
int bml_matrix_dimension_t::N_rows
```

The number of rows.

### 12.2.2.5 NB

```
int bml_matrix_dimension_t::NB
```

The number of blocks/row (or column).

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

- /tmp/bml/src/C-interface/[bml\\_types.h](#)

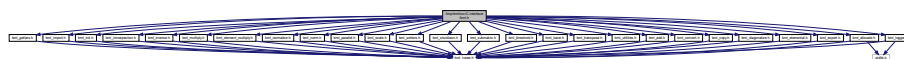
## Chapter 13

# File Documentation

### 13.1 /tmp/bml/src/C-interface/bml.h File Reference

```
#include "bml_add.h"
#include "bml_allocate.h"
#include "bml_convert.h"
#include "bml_copy.h"
#include "bml_diagonalize.h"
#include "bml_elemental.h"
#include "bml_export.h"
#include "bml_getters.h"
#include "bml_import.h"
#include "bml_init.h"
#include "bml_introspection.h"
#include "bml_inverse.h"
#include "bml_logger.h"
#include "bml_multiply.h"
#include "bml_element_multiply.h"
#include "bml_normalize.h"
#include "bml_norm.h"
#include "bml_parallel.h"
#include "bml_scale.h"
#include "bml_setters.h"
#include "bml_shutdown.h"
#include "bml_submatrix.h"
#include "bml_threshold.h"
#include "bml_trace.h"
#include "bml_transpose.h"
#include "bml_utilities.h"
```

Include dependency graph for bml.h:



#### 13.1.1 Detailed Description

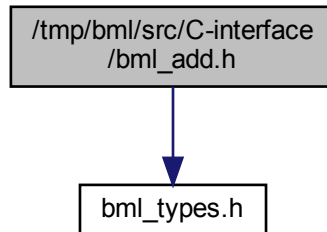
Copyright

Los Alamos National Laboratory 2015

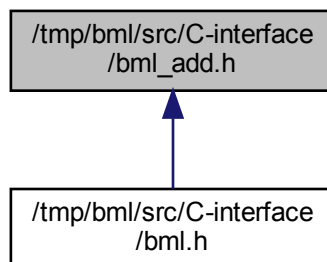
## 13.2 /tmp/bml/src/C-interface/bml\_add.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_add.h:



This graph shows which files directly or indirectly include this file:



### Functions

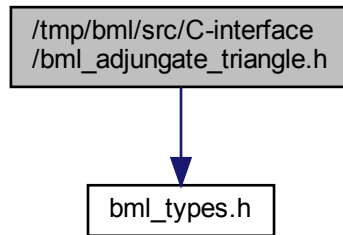
- void `bml_add` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, double alpha, double beta, double threshold)
- double `bml_add_norm` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, double alpha, double beta, double threshold)
- void `bml_add_identity` (`bml_matrix_t` \*A, double beta, double threshold)
- void `bml_scale_add_identity` (`bml_matrix_t` \*A, double alpha, double beta, double threshold)

## 13.3 /tmp/bml/src/C-interface/bml\_adjungate\_triangle.h File Reference

```
#include "bml_types.h"
```



Include dependency graph for bml\_adjungate\_triangle.h:



## Functions

- void `bml_adjungate_triangle` (`bml_matrix_t` \*A, char \*triangle)

### 13.3.1 Function Documentation

#### 13.3.1.1 bml\_adjungate\_triangle()

```
void bml_adjungate_triangle (
    bml_matrix_t * A,
    char * triangle )
```

Adjungates (conjugate transpose) a triangle of a matrix in place.

#### Parameters

in, out	<i>A</i>	The matrix for which the triangle should be adjungated
in	<i>triangle</i>	Which triangle to adjungate ('u': upper, 'l': lower)

Here is the call graph for this function:

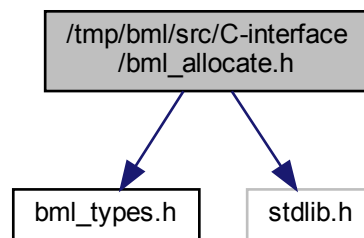


## 13.4 /tmp/bml/src/C-interface/bml\_allocate.h File Reference

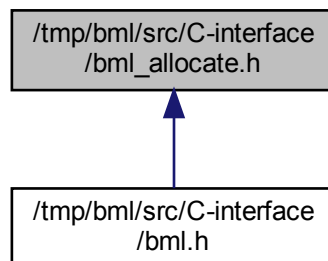
```
#include "bml_types.h"
```

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

Include dependency graph for bml\_allocate.h:



This graph shows which files directly or indirectly include this file:



## Functions

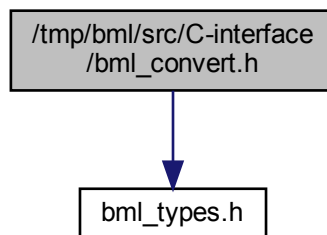
- `int bml_allocated (bml_matrix_t *A)`
- `void * bml_allocate_memory (size_t s)`
- `void * bml_noinit_allocate_memory (size_t s)`
- `void * bml_reallocate_memory (void *ptr, const size_t size)`
- `void bml_free_memory (void *ptr)`
- `void bml_free_ptr (void **ptr)`
- `void bml_deallocate (bml_matrix_t **A)`
- `void bml_deallocate_domain (bml_domain_t *D)`
- `void bml_clear (bml_matrix_t *A)`
- `bml_matrix_t * bml_noinit_rectangular_matrix (bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, bml_matrix_dimension_t matrix_dimension, bml_distribution_mode_t distrib_mode)`

- `bml_matrix_t * bml_noinit_matrix` (`bml_matrix_type_t` matrix\_type, `bml_matrix_precision_t` matrix\_precision, int N, int M, `bml_distribution_mode_t` distrib\_mode)
- `bml_matrix_t * bml_zero_matrix` (`bml_matrix_type_t` matrix\_type, `bml_matrix_precision_t` matrix\_precision, int N, int M, `bml_distribution_mode_t` distrib\_mode)
- `bml_matrix_t * bml_random_matrix` (`bml_matrix_type_t` matrix\_type, `bml_matrix_precision_t` matrix\_precision, int N, int M, `bml_distribution_mode_t` distrib\_mode)
- `bml_matrix_t * bml_banded_matrix` (`bml_matrix_type_t` matrix\_type, `bml_matrix_precision_t` matrix\_precision, int N, int M, `bml_distribution_mode_t` distrib\_mode)
- `bml_matrix_t * bml_identity_matrix` (`bml_matrix_type_t` matrix\_type, `bml_matrix_precision_t` matrix\_precision, int N, int M, `bml_distribution_mode_t` distrib\_mode)
- `bml_domain_t * bml_default_domain` (int N, int M, `bml_distribution_mode_t` distrib\_mode)
- void `bml_update_domain` (`bml_matrix_t` \*A, int \*localPartMin, int \*localPartMax, int \*nnodesInPart)

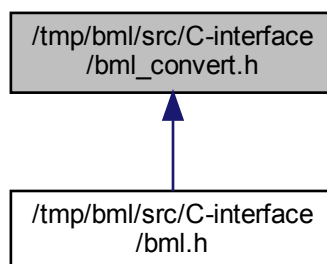
## 13.5 /tmp/bml/src/C-interface/bml\_convert.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_convert.h:



This graph shows which files directly or indirectly include this file:



## Functions

- `bml_matrix_t * bml_convert (bml_matrix_t *A, bml_matrix_type_t matrix_type, bml_matrix_precision_t matrix_precision, int M, bml_distribution_mode_t distrib_mode)`

### 13.5.1 Function Documentation

#### 13.5.1.1 `bml_convert()`

```
bml_matrix_t* bml_convert (
    bml_matrix_t * A,
    bml_matrix_type_t matrix_type,
    bml_matrix_precision_t matrix_precision,
    int M,
    bml_distribution_mode_t distrib_mode )
```

Convert a bml matrix to another type.

$A \rightarrow B$

#### Parameters

<i>A</i>	The input matrix.
----------	-------------------

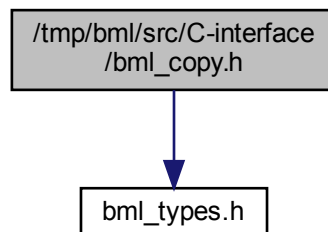
#### Returns

The converted matrix *B*.

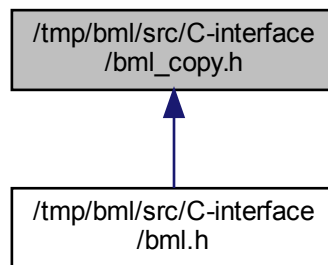
## 13.6 `/tmp/bml/src/C-interface/bml_copy.h` File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_copy.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- `bml_matrix_t * bml_copy_new (bml_matrix_t *A)`
- `void bml_copy (bml_matrix_t *A, bml_matrix_t *B)`
- `void bml_reorder (bml_matrix_t *A, int *perm)`
- `void bml_copy_domain (bml_domain_t *A, bml_domain_t *B)`
- `void bml_save_domain (bml_matrix_t *A)`
- `void bml_restore_domain (bml_matrix_t *A)`

### 13.6.1 Function Documentation

#### 13.6.1.1 `bml_copy()`

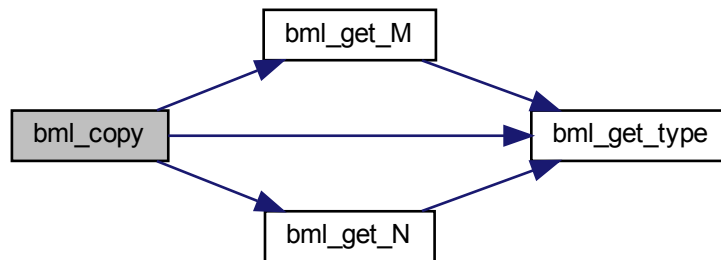
```
void bml_copy (
    bml_matrix_t * A,
    bml_matrix_t * B )
```

Copy a matrix.

##### Parameters

<i>A</i>	Matrix to copy
<i>B</i>	Copy of Matrix A

Here is the call graph for this function:



### 13.6.1.2 bml\_copy\_domain()

```
void bml_copy_domain (
    bml_domain_t * A,
    bml_domain_t * B )
```

Copy a domain.

#### Parameters

<i>A</i>	Domain to copy
<i>B</i>	Copy of Domain A

Here is the call graph for this function:



### 13.6.1.3 bml\_copy\_new()

```
bml_matrix_t* bml_copy_new (
    bml_matrix_t * A )
```

Copy a matrix - result is a new matrix.

## Parameters

<i>A</i>	Matrix to copy
----------	----------------

## Returns

A Copy of A

Here is the call graph for this function:



#### 13.6.1.4 bml\_reorder()

```
void bml_reorder (
    bml_matrix_t * A,
    int * perm )
```

Reorder a matrix in place.

## Parameters

<i>A</i>	Matrix to reorder
<i>perm</i>	permutation vector for reordering

Here is the call graph for this function:



### 13.6.1.5 bml\_restore\_domain()

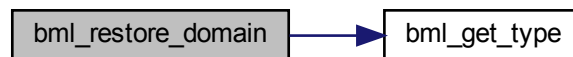
```
void bml_restore_domain (
    bml_matrix_t * A )
```

Restore to saved domain for bml matrix.

#### Parameters

<i>A</i>	Matrix with domain
----------	--------------------

Here is the call graph for this function:



### 13.6.1.6 bml\_save\_domain()

```
void bml_save_domain (
    bml_matrix_t * A )
```

Save current domain for bml matrix.

#### Parameters

<i>A</i>	Matrix with domain
----------	--------------------

Here is the call graph for this function:

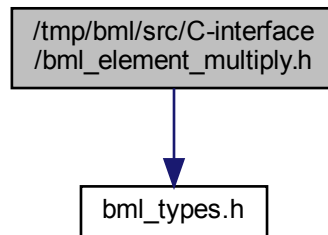


## 13.7 /tmp/bml/src/C-interface/bml\_element\_multiply.h File Reference

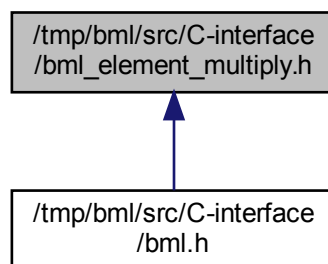
```
#include "bml_types.h"
```



Include dependency graph for bml\_element\_multiply.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void `bml_element_multiply_AB` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, `bml_matrix_t` \*C, double threshold)

### 13.7.1 Function Documentation

#### 13.7.1.1 bml\_element\_multiply\_AB()

```

void bml_element_multiply_AB (
    bml_matrix_t * A,
    bml_matrix_t * B,
    bml_matrix_t * C,
    double threshold )
  
```

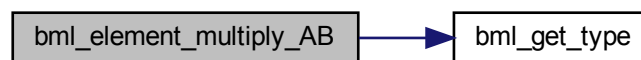
Element-wise Matrix multiply (Hadamard product)

$$C_{ij} \leftarrow A_{ij} * B_{ij}$$

## Parameters

<i>A</i>	Matrix A
<i>B</i>	Matrix B
<i>C</i>	Matrix C
<i>threshold</i>	Threshold for multiplication

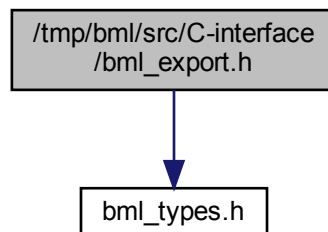
Here is the call graph for this function:



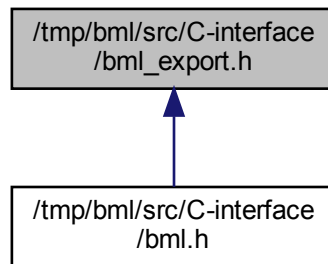
## 13.8 /tmp/bml/src/C-interface/bml\_export.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_export.h`:



This graph shows which files directly or indirectly include this file:



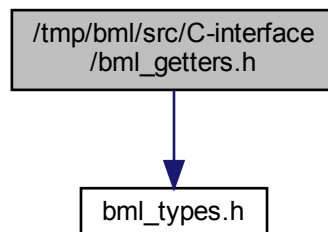
## Functions

- void \* [bml\\_export\\_to\\_dense](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_dense\\_order\\_t](#) order)

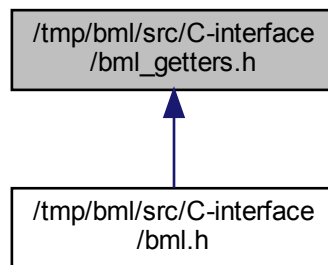
## 13.9 /tmp/bml/src/C-interface/bml\_getters.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_getters.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- void \* [bml\\_get\\_element](#) ([bml\\_matrix\\_t](#) \*A, int i, int j)
- void \* [bml\\_get\\_row](#) ([bml\\_matrix\\_t](#) \*A, int i)
- void \* [bml\\_get\\_diagonal](#) ([bml\\_matrix\\_t](#) \*A)

### 13.9.1 Function Documentation

#### 13.9.1.1 [bml\\_get\\_diagonal\(\)](#)

```
void* bml_get_diagonal (  
    bml\_matrix\_t * A )
```

Get the diagonal.

##### Parameters

<i>A</i>	The matrix.
----------	-------------

**Returns**

The diagonal (an array)

Here is the call graph for this function:

**13.9.1.2 bml\_get\_element()**

```
void* bml_get_element (
    bml_matrix_t * A,
    int i,
    int j )
```

Return a single matrix element.

**Parameters**

<i>i</i>	The row index
<i>j</i>	The column index
<i>A</i>	The bml matrix

**Returns**

The matrix element

Here is the call graph for this function:



### 13.9.1.3 bml\_get\_row()

```
void* bml_get_row (
    bml_matrix_t * A,
    int i )
```

Get a whole row.

#### Parameters

<i>A</i>	The matrix.
<i>i</i>	The row index.

#### Returns

An array (needs to be cast into the appropriate type).

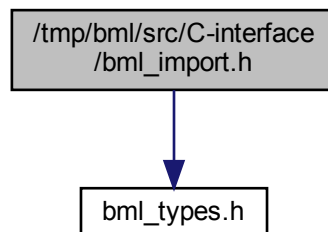
Here is the call graph for this function:



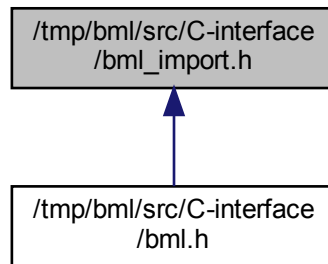
## 13.10 /tmp/bml/src/C-interface/bml\_import.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_import.h:



This graph shows which files directly or indirectly include this file:



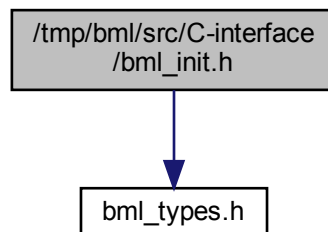
## Functions

- `bml_matrix_t * bml_import_from_dense` (`bml_matrix_type_t` matrix\_type, `bml_matrix_precision_t` matrix\_precision, `bml_dense_order_t` order, int N, int M, void \*A, double threshold, `bml_distribution_mode_t` distribution\_mode)

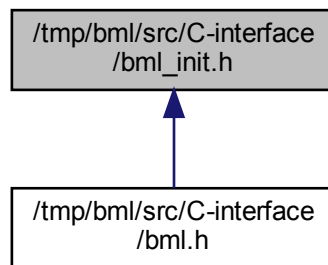
## 13.11 /tmp/bml/src/C-interface/bml\_init.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_init.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- void `bml_init` ( )
- void `bml_initF` (int *fcomm*)

### 13.11.1 Function Documentation

#### 13.11.1.1 `bml_init()`

```
void bml_init ( )
```

Initialize.

##### Parameters

<i>argc</i>	Number of args
<i>argv</i>	Args

#### 13.11.1.2 `bml_initF()`

```
void bml_initF (
    int fcomm )
```

Initialize from Fortran.



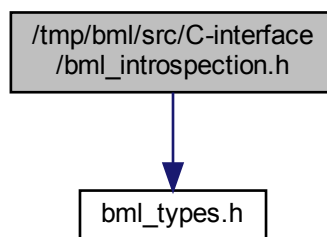
## Parameters

<i>Comm</i>	from Fortran
-------------	--------------

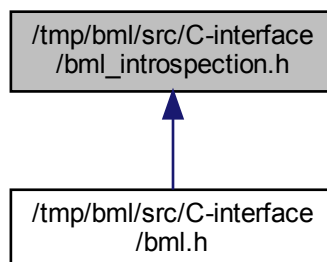
## 13.12 /tmp/bml/src/C-interface/bml\_introspection.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_introspection.h:



This graph shows which files directly or indirectly include this file:



## Functions

- [bml\\_matrix\\_type\\_t bml\\_get\\_type \(bml\\_matrix\\_t \\*A\)](#)
- [bml\\_matrix\\_type\\_t bml\\_get\\_deep\\_type \(bml\\_matrix\\_t \\*A\)](#)
- [bml\\_matrix\\_precision\\_t bml\\_get\\_precision \(bml\\_matrix\\_t \\*A\)](#)
- [int bml\\_get\\_N \(bml\\_matrix\\_t \\*A\)](#)
- [int bml\\_get\\_M \(bml\\_matrix\\_t \\*A\)](#)
- [int bml\\_get\\_NB \(bml\\_matrix\\_t \\*A\)](#)

- `int bml_get_row_bandwidth (bml_matrix_t *A, int i)`
- `int bml_get_bandwidth (bml_matrix_t *A)`
- `double bml_get_sparsity (bml_matrix_t *A, double threshold)`
- `bml_distribution_mode_t bml_get_distribution_mode (bml_matrix_t *A)`
- `bml_matrix_t * bml_get_local_matrix (bml_matrix_t *A)`
- `void * bml_get_data_ptr (bml_matrix_t *A)`

### 13.12.1 Function Documentation

#### 13.12.1.1 `bml_get_bandwidth()`

```
int bml_get_bandwidth (
    bml_matrix_t * A )
```

Return the bandwidth of a matrix.

##### Parameters

<i>A</i>	The bml matrix.
----------	-----------------

##### Returns

The bandwidth of row *i*.

Here is the call graph for this function:



#### 13.12.1.2 `bml_get_deep_type()`

```
bml_matrix_type_t bml_get_deep_type (
    bml_matrix_t * A )
```

Return the matrix type for the data storage For distributed2 matrices, return the matrix type for the local submatrices

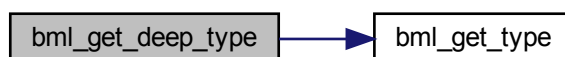
**Parameters**

<i>A</i>	The matrix.
----------	-------------

**Returns**

The matrix type

Here is the call graph for this function:

**13.12.1.3 bml\_get\_distribution\_mode()**

```
bml_distribution_mode_t bml_get_distribution_mode (  
    bml_matrix_t * A )
```

Return the distribution mode of a matrix.

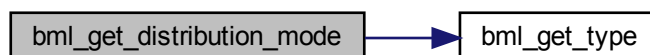
**Parameters**

<i>A</i>	The bml matrix.
----------	-----------------

**Returns**

The distribution mode of matrix A.

Here is the call graph for this function:



#### 13.12.1.4 bml\_get\_M()

```
int bml_get_M (  
    bml_matrix_t * A )
```

Return the matrix parameter M.

##### Parameters

<i>A</i>	The matrix.
----------	-------------

##### Returns

The matrix parameter M.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 13.12.1.5 bml\_get\_N()

```
int bml_get_N (  
    bml_matrix_t * A )
```

Return the matrix size.

##### Parameters

<i>A</i>	The matrix.
----------	-------------

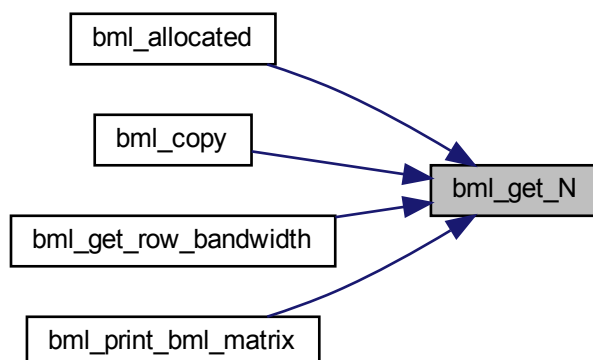
#### Returns

The matrix size.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 13.12.1.6 bml\_get\_precision()

```
bml_matrix_precision_t bml_get_precision (  
    bml_matrix_t * A )
```

Return the matrix precision.

#### Parameters

<i>A</i>	The matrix.
----------	-------------

**Returns**

The matrix precision.

Here is the call graph for this function:



Here is the caller graph for this function:

**13.12.1.7 bml\_get\_row\_bandwidth()**

```

int bml_get_row_bandwidth (
    bml_matrix_t * A,
    int i )
  
```

Return the bandwidth of a row in the matrix.

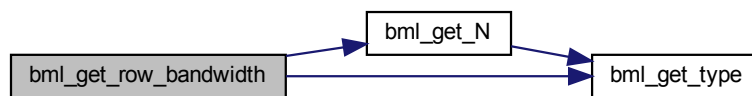
**Parameters**

<i>A</i>	The bml matrix.
<i>i</i>	The row index.

**Returns**

The bandwidth of row *i*.

Here is the call graph for this function:

**13.12.1.8 bml\_get\_sparsity()**

```
double bml_get_sparsity (
    bml_matrix_t * A,
    double threshold )
```

Return the sparsity of a matrix.

**Parameters**

<i>A</i>	The bml matrix.
<i>threshold</i>	The threshold used to compute the sparsity.

**Returns**

The sparsity of matrix *A*.

Here is the call graph for this function:



### 13.12.1.9 bml\_get\_type()

```
bml_matrix_type_t bml_get_type (  
    bml_matrix_t * A )
```

Returns the matrix type.

If the matrix is not initialized yet, a type of "uninitialized" is returned.

#### Parameters

<i>A</i>	The matrix.
----------	-------------

#### Returns

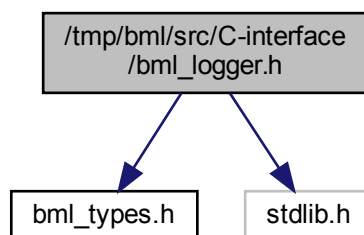
The matrix type.

## 13.13 /tmp/bml/src/C-interface/bml\_logger.h File Reference

```
#include "bml_types.h"
```

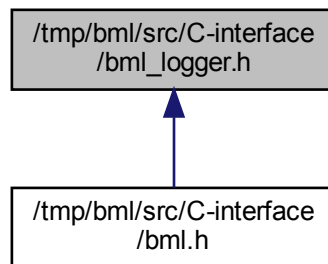
```
#include <stdlib.h>
```

Include dependency graph for bml\_logger.h:





This graph shows which files directly or indirectly include this file:



## Macros

- `#define LOG_DEBUG(format, ...) bml_log_location(BML_LOG_DEBUG, __FILE__, __LINE__, format, ##__VA_ARGS__)`
- `#define LOG_INFO(format, ...) bml_log(BML_LOG_INFO, format, ##__VA_ARGS__)`
- `#define LOG_WARN(format, ...) bml_log_location(BML_LOG_WARNING, __FILE__, __LINE__, format, ##__VA_ARGS__)`
- `#define LOG_ERROR(format, ...) bml_log_location(BML_LOG_ERROR, __FILE__, __LINE__, format, ##__VA_ARGS__)`

## Enumerations

- `enum bml_log_level_t { BML_LOG_DEBUG, BML_LOG_INFO, BML_LOG_WARNING, BML_LOG_ERROR }`

## Functions

- `void bml_log(bml_log_level_t log_level, char *format,...)`
- `void bml_log_location(bml_log_level_t log_level, char *filename, int linenum, char *format,...)`
- `char * bml_version(void)`

*Return version string of library.*

### 13.13.1 Macro Definition Documentation

#### 13.13.1.1 LOG\_DEBUG

```

#define LOG_DEBUG(
    format,
    ... ) bml_log_location(BML_LOG_DEBUG, __FILE__, __LINE__, format, ##__VA_ARGS__
)
  
```

Convenience macro to write a BML\_LOG\_DEBUG level message.

### 13.13.1.2 LOG\_ERROR

```
#define LOG_ERROR(  
    format,  
    ... ) bml_log_location(BML_LOG_ERROR, __FILE__, __LINE__, format, ##__VA_ARGS__  
_)
```

Convenience macro to write a BML\_LOG\_ERROR level message.

### 13.13.1.3 LOG\_INFO

```
#define LOG_INFO(  
    format,  
    ... ) bml_log(BML_LOG_INFO, format, ##__VA_ARGS__)
```

Convenience macro to write a BML\_LOG\_INFO level message.

### 13.13.1.4 LOG\_WARN

```
#define LOG_WARN(  
    format,  
    ... ) bml_log_location(BML_LOG_WARNING, __FILE__, __LINE__, format, ##__VA_ARGS__  
S_)
```

Convenience macro to write a BML\_LOG\_WARNING level message.

## 13.13.2 Enumeration Type Documentation

### 13.13.2.1 bml\_log\_level\_t

```
enum bml_log_level_t
```

The log-levels.

Enumerator

BML_LOG_DEBUG	Debugging messages.
BML_LOG_INFO	Info messages.
BML_LOG_WARNING	Warning messages.
BML_LOG_ERROR	Error messages.

## 13.13.3 Function Documentation

### 13.13.3.1 bml\_log()

```
void bml_log (
    bml_log_level_t log_level,
    char * format,
    ... )
```

Log a message.

#### Parameters

<i>log_level</i>	The log level.
<i>format</i>	The format (as in printf()).

### 13.13.3.2 bml\_log\_location()

```
void bml_log_location (
    bml_log_level_t log_level,
    char * filename,
    int linenumber,
    char * format,
    ... )
```

Log a message with location, i.e. filename and linenumber..

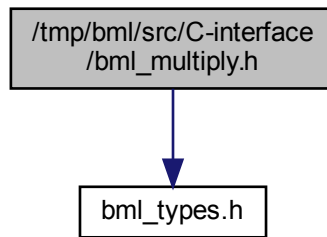
#### Parameters

<i>log_level</i>	The log level.
<i>filename</i>	The filename to log.
<i>linenumber</i>	The linenumber.
<i>format</i>	The format (as in printf()).

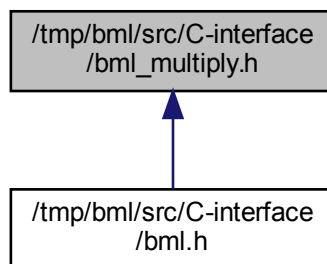
## 13.14 /tmp/bml/src/C-interface/bml\_multiply.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_multiply.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- void `bml_multiply` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, `bml_matrix_t` \*C, double alpha, double beta, double threshold)
- void \* `bml_multiply_x2` (`bml_matrix_t` \*X, `bml_matrix_t` \*X2, double threshold)
- void `bml_multiply_AB` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, `bml_matrix_t` \*C, double threshold)
- void `bml_multiply_adjust_AB` (`bml_matrix_t` \*A, `bml_matrix_t` \*B, `bml_matrix_t` \*C, double threshold)

### 13.14.1 Function Documentation

### 13.14.1.1 bml\_multiply()

```
void bml_multiply (
    bml_matrix_t * A,
    bml_matrix_t * B,
    bml_matrix_t * C,
    double alpha,
    double beta,
    double threshold )
```

Matrix multiply.

$$C \leftarrow \alpha A B + \beta C$$

#### Parameters

<i>A</i>	Matrix A
<i>B</i>	Matrix B
<i>C</i>	Matrix C
<i>alpha</i>	Scalar factor that multiplies A * B
<i>beta</i>	Scalar factor that multiplies C
<i>threshold</i>	Threshold for multiplication

Here is the call graph for this function:



### 13.14.1.2 bml\_multiply\_AB()

```
void bml_multiply_AB (
    bml_matrix_t * A,
    bml_matrix_t * B,
    bml_matrix_t * C,
    double threshold )
```

Matrix multiply.

$$C = A * B$$

#### Parameters

<i>A</i>	Matrix A
<i>B</i>	Matrix B
<i>C</i>	Matrix C
<i>threshold</i>	Threshold for multiplication

Here is the call graph for this function:



### 13.14.1.3 bml\_multiply\_adjust\_AB()

```

void bml_multiply_adjust_AB (
    bml_matrix_t * A,
    bml_matrix_t * B,
    bml_matrix_t * C,
    double threshold )
  
```

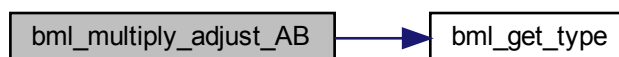
Matrix multiply with threshold adjustment.

$C = A * B$

Parameters

<i>A</i>	Matrix A
<i>B</i>	Matrix B
<i>C</i>	Matrix C
<i>threshold</i>	Threshold for multiplication

Here is the call graph for this function:



### 13.14.1.4 bml\_multiply\_x2()

```

void* bml_multiply_x2 (
    bml_matrix_t * X,
  
```

```

    bml_matrix_t * X2,
    double threshold )

```

Matrix multiply.

$$X^2 \leftarrow X X$$

Parameters

<i>X</i>	Matrix X
<i>X2</i>	MatrixX2
<i>threshold</i>	Threshold for multiplication

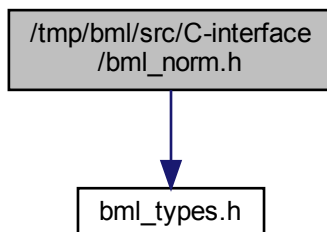
Here is the call graph for this function:



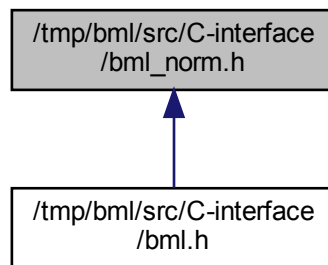
## 13.15 /tmp/bml/src/C-interface/bml\_norm.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_norm.h:



This graph shows which files directly or indirectly include this file:



## Functions

- double [bml\\_sum\\_squares](#) ([bml\\_matrix\\_t](#) \*A)
- double [bml\\_sum\\_squares2](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_matrix\\_t](#) \*B, double alpha, double beta, double threshold)
- double [bml\\_sum\\_AB](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_matrix\\_t](#) \*B, double alpha, double threshold)
- double [bml\\_sum\\_squares\\_submatrix](#) ([bml\\_matrix\\_t](#) \*A, int core\_size)
- double [bml\\_fnorm](#) ([bml\\_matrix\\_t](#) \*A)
- double [bml\\_fnorm2](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_matrix\\_t](#) \*B)

### 13.15.1 Function Documentation

#### 13.15.1.1 [bml\\_fnorm\(\)](#)

```
double bml_fnorm (
    bml\_matrix\_t * A )
```

Calculate the Frobenius norm of a matrix.

##### Parameters

<a href="#">A</a>	Matrix A
-------------------	----------



**Returns**

Frobenius norm of Matrix A

Here is the call graph for this function:

**13.15.1.2 bml\_fnorm2()**

```
double bml_fnorm2 (  
    bml_matrix_t * A,  
    bml_matrix_t * B )
```

Calculate the Frobenius norm of 2 matrices.

**Parameters**

<i>A</i>	Matrix A
<i>B</i>	Matrix B

**Returns**

Frobenius norm of Matrix A

Here is the call graph for this function:



### 13.15.1.3 bml\_sum\_AB()

```
double bml_sum_AB (
    bml_matrix_t * A,
    bml_matrix_t * B,
    double alpha,
    double threshold )
```

Calculate sum of all the elements of  $\alpha A(i,j) * B(i,j)$

#### Parameters

<i>A</i>	Matrix
<i>B</i>	Matrix
<i>alpha</i>	Multiplier for matrix A
<i>threshold</i>	Threshold

#### Returns

sum of squares of  $\alpha * A + \beta * B$

Here is the call graph for this function:



### 13.15.1.4 bml\_sum\_squares()

```
double bml_sum_squares (
    bml_matrix_t * A )
```

Calculate the sum of squares of all the elements of a matrix.

#### Parameters

<i>A</i>	Matrix A
----------	----------

#### Returns

sum of squares of all elements in A

Here is the call graph for this function:



### 13.15.1.5 bml\_sum\_squares2()

```
double bml_sum_squares2 (
    bml_matrix_t * A,
    bml_matrix_t * B,
    double alpha,
    double beta,
    double threshold )
```

Calculate sum of squares of all the elements of  $\alpha A + \beta B$

#### Parameters

<i>A</i>	Matrix
<i>B</i>	Matrix
<i>alpha</i>	Multiplier for matrix A
<i>beta</i>	Multiplier for matrix B
<i>threshold</i>	Threshold

#### Returns

sum of squares of  $\alpha * A + \beta * B$

Here is the call graph for this function:



### 13.15.1.6 bml\_sum\_squares\_submatrix()

```
double bml_sum_squares_submatrix (
    bml_matrix_t * A,
    int core_size )
```

Calculate the sum of squares of all the elements of a matrix.

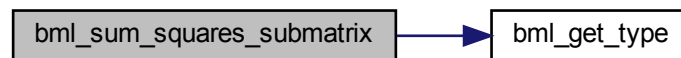
#### Parameters

<i>A</i>	Matrix A
<i>core_pos</i>	Core rows in A
<i>core_size</i>	Number of core rows

#### Returns

sum of squares of all elements in A

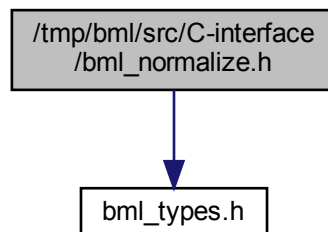
Here is the call graph for this function:



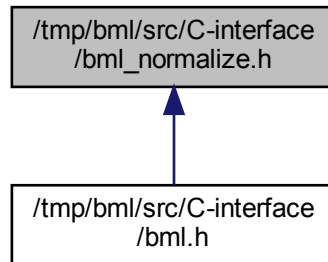
## 13.16 /tmp/bml/src/C-interface/bml\_normalize.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_normalize.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [bml\\_normalize](#) ([bml\\_matrix\\_t](#) \*A, double mineval, double maxeval)
- void \* [bml\\_gershgorin](#) ([bml\\_matrix\\_t](#) \*A)
- void \* [bml\\_gershgorin\\_partial](#) ([bml\\_matrix\\_t](#) \*A, int nrows)
- void \* [bml\\_accumulate\\_offdiag](#) ([bml\\_matrix\\_t](#) \*A, int flag)

### 13.16.1 Function Documentation

#### 13.16.1.1 bml\_gershgorin()

```
void* bml_gershgorin (
    bml\_matrix\_t * A )
```

Calculate Gershgorin bounds.

##### Parameters

A	Matrix to scale returns mineval Calculated min value returns maxeval Calculated max value
---	---

Here is the call graph for this function:



### 13.16.1.2 bml\_gershgorin\_partial()

```
void* bml_gershgorin_partial (
    bml_matrix_t * A,
    int nrows )
```

Calculate Gershgorin bounds for partial matrix.

#### Parameters

<i>A</i>	Matrix to scale
<i>nrows</i>	Number of rows used returns mineval Calculated min value returns maxeval Calculated max value

Here is the call graph for this function:



### 13.16.1.3 bml\_normalize()

```
void bml_normalize (
    bml_matrix_t * A,
    double mineval,
    double maxeval )
```

Normalize matrix given Gershgorin bounds.

#### Parameters

<i>A</i>	Matrix to scale
<i>mineval</i>	Calculated min value
<i>maxeval</i>	Calculated max value

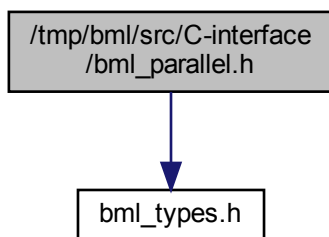
Here is the call graph for this function:



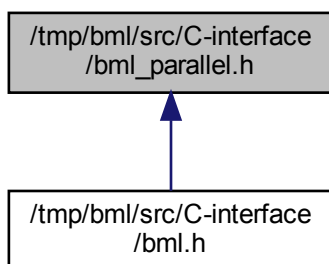
## 13.17 /tmp/bml/src/C-interface/bml\_parallel.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_parallel.h:



This graph shows which files directly or indirectly include this file:



## Functions

- int [bml\\_getNRanks](#) (void)
- int [bml\\_getMyRank](#) (void)
- void **bml\_initParallelF** (int fcomm)
- void **bml\_shutdownParallelF** ()
- int **bml\_printRank** (void)
- void **bml\_shutdownParallel** (void)
- void **bml\_barrierParallel** (void)
- void **bml\_sumRealReduce** (double \*value)
- void **bml\_minRealReduce** (double \*value)
- void **bml\_maxRealReduce** (double \*value)
- void [bml\\_allGatherVParallel](#) ([bml\\_matrix\\_t](#) \*A)

### 13.17.1 Function Documentation

#### 13.17.1.1 [bml\\_allGatherVParallel\(\)](#)

```
void bml_allGatherVParallel (
    bml\_matrix\_t * A )
```

Exchange pieces of matrix across MPI ranks.

##### Parameters

<i>A</i>	Matrix A
----------	----------

Here is the call graph for this function:

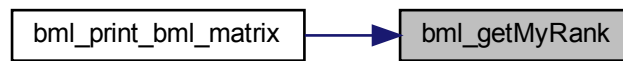


#### 13.17.1.2 [bml\\_getMyRank\(\)](#)

```
int bml_getMyRank (
    void )
```



Get local MPI rank. Here is the caller graph for this function:



### 13.17.1.3 bml\_getNRanks()

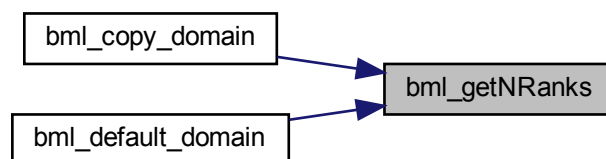
```
int bml_getNRanks (
    void )
```

Initialize.

Parameters

<i>argc</i>	Number of args
<i>argv</i>	Args Get number of MPI ranks.

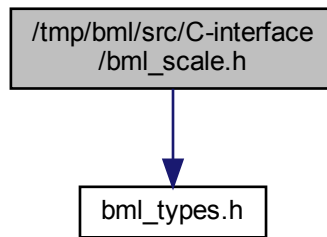
Here is the caller graph for this function:



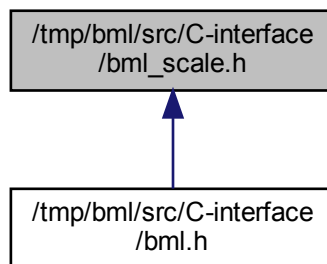
## 13.18 /tmp/bml/src/C-interface/bml\_scale.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_scale.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- `bml_matrix_t * bml_scale_new` (`void *scale_factor`, `bml_matrix_t *A`)
- `void bml_scale` (`void *scale_factor`, `bml_matrix_t *A`, `bml_matrix_t *B`)
- `void bml_scale_inplace` (`void *scale_factor`, `bml_matrix_t *A`)

### 13.18.1 Function Documentation

#### 13.18.1.1 `bml_scale()`

```
void bml_scale (  
    void * scale_factor,  
    bml_matrix_t * A,  
    bml_matrix_t * B )
```

Scale a matrix - resulting matrix exists.

## Parameters

<i>scale_factor</i>	Scale factor for A
<i>A</i>	Matrix to scale
<i>B</i>	Scaled Matrix

Here is the call graph for this function:



### 13.18.1.2 bml\_scale\_inplace()

```
void bml_scale_inplace (  
    void * scale_factor,  
    bml_matrix_t * A )
```

Scale a matrix in place, i.e. the matrix is overwritten.

## Parameters

<i>scale_factor</i>	Scale factor for A
<i>A</i>	[inout] Matrix to scale

Here is the call graph for this function:



### 13.18.1.3 bml\_scale\_new()

```
bml_matrix_t* bml_scale_new (  
    void * scale_factor,  
    bml_matrix_t * A )
```

Scale a matrix - resulting matrix is new.

#### Parameters

<i>scale_factor</i>	Scale factor for A
<i>A</i>	Matrix to scale

#### Returns

A Scaled Copy of A

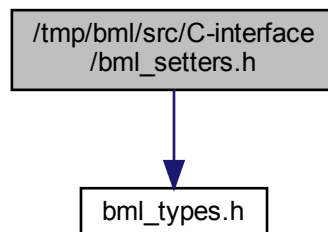
Here is the call graph for this function:



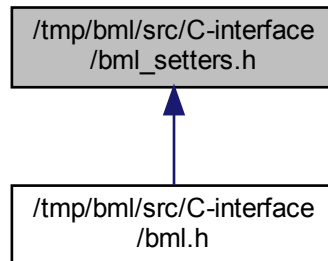
## 13.19 /tmp/bml/src/C-interface/bml\_setters.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_setters.h`:



This graph shows which files directly or indirectly include this file:



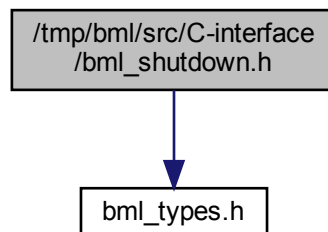
## Functions

- void **bml\_set\_element\_new** ([bml\\_matrix\\_t](#) \*A, int i, int j, void \*value)
- void **bml\_set\_element** ([bml\\_matrix\\_t](#) \*A, int i, int j, void \*value)
- void **bml\_set\_row** ([bml\\_matrix\\_t](#) \*A, int i, void \*row, double threshold)
- void **bml\_set\_diagonal** ([bml\\_matrix\\_t](#) \*A, void \*diagonal, double threshold)

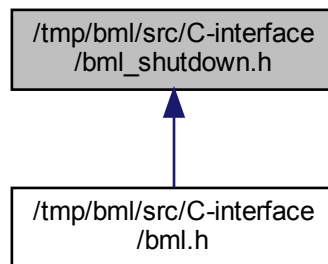
## 13.20 /tmp/bml/src/C-interface/bml\_shutdown.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for `bml_shutdown.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- void [bml\\_shutdown](#) ()
- void [bml\\_shutdownF](#) ()

### 13.20.1 Function Documentation

#### 13.20.1.1 `bml_shutdown()`

```
void bml_shutdown ( )
```

Shutdown.

#### 13.20.1.2 `bml_shutdownF()`

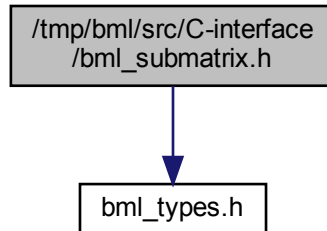
```
void bml_shutdownF ( )
```

Shutdown from Fortran.

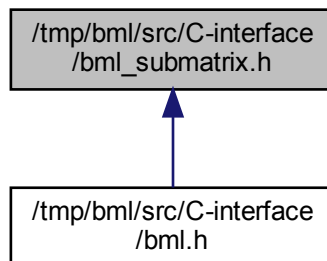
## 13.21 /tmp/bml/src/C-interface/bml\_submatrix.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_submatrix.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [bml\\_matrix2submatrix\\_index](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_matrix\\_t](#) \*B, int \*nodelist, int nsize, int \*core\_halo\_index, int \*vsize, int double\_jump\_flag)
- void [bml\\_matrix2submatrix\\_index\\_graph](#) ([bml\\_matrix\\_t](#) \*B, int \*nodelist, int nsize, int \*core\_halo\_index, int \*vsize, int double\_jump\_flag)
- void [bml\\_matrix2submatrix](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_matrix\\_t](#) \*B, int \*core\_halo\_index, int lsize)
- void [bml\\_submatrix2matrix](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_matrix\\_t](#) \*B, int \*core\_halo\_index, int lsize, int lsize, double threshold)
- void [bml\\_adjacency](#) ([bml\\_matrix\\_t](#) \*A, int \*xadj, int \*adjncy, int base\_flag)
- void [bml\\_adjacency\\_group](#) ([bml\\_matrix\\_t](#) \*A, int \*hindex, int nnodes, int \*xadj, int \*adjncy, int base\_flag)
- [bml\\_matrix\\_t](#) \* [bml\\_group\\_matrix](#) ([bml\\_matrix\\_t](#) \*A, int \*hindex, int ngroups, double threshold)
- [bml\\_matrix\\_t](#) \* [bml\\_extract\\_submatrix](#) ([bml\\_matrix\\_t](#) \*A, int irow, int icol, int B\_N, int B\_M)
- void [bml\\_assign\\_submatrix](#) ([bml\\_matrix\\_t](#) \*A, [bml\\_matrix\\_t](#) \*B, int irow, int icol)

## 13.21.1 Function Documentation

### 13.21.1.1 bml\_adjacency()

```
void bml_adjacency (
    bml_matrix_t * A,
    int * xadj,
    int * adjncy,
    int base_flag )
```

Assemble adjacency structures from matrix based on rows.

#### Parameters

<i>A</i>	Submatrix A
<i>xadj</i>	index to start of each row
<i>adjncy</i>	adjacency vector
<i>base_flag</i>	to return 0- or 1-based

Here is the call graph for this function:



### 13.21.1.2 bml\_adjacency\_group()

```
void bml_adjacency_group (
    bml_matrix_t * A,
    int * hindex,
    int nnodes,
    int * xadj,
    int * adjncy,
    int base_flag )
```

Assemble adjacency structures from matrix based on groups of rows.

#### Parameters

<i>A</i>	Submatrix A
<i>hindex</i>	Index for each node element



## Parameters

<i>nnodes</i>	Number of groups
<i>xadj</i>	index to start of each row
<i>adjncy</i>	adjacency vector
<i>base_flag</i>	return 0- or 1-based

Here is the call graph for this function:



## 13.21.1.3 bml\_group\_matrix()

```

bml_matrix_t* bml_group_matrix (
    bml_matrix_t * A,
    int * hindex,
    int ngroups,
    double threshold )
  
```

Assemble matrix based on groups of rows from a matrix.

## Parameters

<i>A</i>	Matrix A
<i>hindex</i>	Indices of nodes
<i>ngroups</i>	Number of groups
<i>threshold</i>	Threshold for graph

Here is the call graph for this function:



#### 13.21.1.4 bml\_matrix2submatrix()

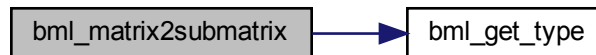
```
void bml_matrix2submatrix (
    bml_matrix_t * A,
    bml_matrix_t * B,
    int * core_halo_index,
    int lsize )
```

Extract a submatrix from a matrix given a set of core+halo rows.

##### Parameters

<i>A</i>	Matrix A
<i>B</i>	Submatrix B
<i>core_halo_index</i>	Set of row indices for submatrix
<i>lsize</i>	Number of indices

Here is the call graph for this function:



#### 13.21.1.5 bml\_matrix2submatrix\_index()

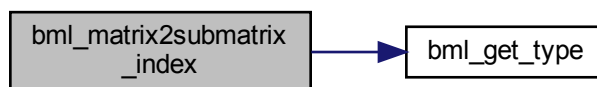
```
void bml_matrix2submatrix_index (
    bml_matrix_t * A,
    bml_matrix_t * B,
    int * nodelist,
    int nsize,
    int * core_halo_index,
    int * vsize,
    int double_jump_flag )
```

Determine element indices for submatrix, given a set of nodes/orbitals.

##### Parameters

<i>A</i>	Hamiltonian matrix A
<i>B</i>	Graph matrix B
<i>nodelist</i>	List of node/orbital indices
<i>nsize</i>	Size of nodelist
<i>core_halo_index</i>	List of core+halo indices
<i>vsize</i>	Size of core_halo_index and core_pos
<i>double_jump_flag</i>	Flag to use double jump (0=no, 1=yes)

Here is the call graph for this function:



### 13.21.1.6 bml\_matrix2submatrix\_index\_graph()

```

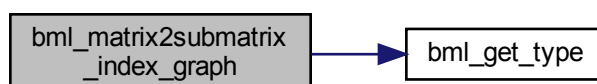
void bml_matrix2submatrix_index_graph (
    bml_matrix_t * B,
    int * nodelist,
    int nsize,
    int * core_halo_index,
    int * vsize,
    int double_jump_flag )
  
```

Determine element indices for submatrix, given a set of nodes/orbitals.

#### Parameters

<i>B</i>	Graph matrix B
<i>nodelist</i>	List of node/orbital indices
<i>nsize</i>	Size of nodelist
<i>core_halo_index</i>	List of core+halo indices
<i>vsize</i>	Size of core_halo_index and core_pos
<i>double_jump_flag</i>	Flag to use double jump (0=no, 1=yes)

Here is the call graph for this function:



### 13.21.1.7 bml\_submatrix2matrix()

```
void bml_submatrix2matrix (
    bml_matrix_t * A,
    bml_matrix_t * B,
    int * core_halo_index,
    int lsize,
    int llsz,
    double threshold )
```

Assemble submatrix into a full matrix based on core+halo indices.

#### Parameters

<i>A</i>	Submatrix A
<i>B</i>	Matrix B
<i>core_halo_index</i>	Set of submatrix row indices
<i>lsize</i>	Number of indices
<i>llsz</i>	Number of core positions

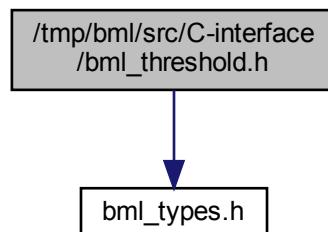
Here is the call graph for this function:



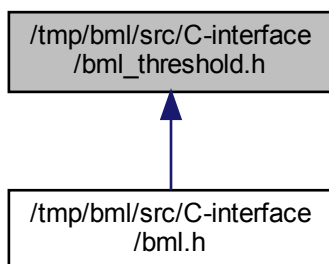
## 13.22 /tmp/bml/src/C-interface/bml\_threshold.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_threshold.h:



This graph shows which files directly or indirectly include this file:



## Functions

- `bml_matrix_t * bml_threshold_new (bml_matrix_t *A, double threshold)`
- `void bml_threshold (bml_matrix_t *A, double threshold)`

### 13.22.1 Function Documentation

#### 13.22.1.1 `bml_threshold()`

```
void bml_threshold (  
    bml_matrix_t * A,  
    double threshold )
```

Threshold matrix.

##### Parameters

<i>A</i>	Matrix to be thresholded
<i>threshold</i>	Threshold value

**Returns**

Thresholded A

Here is the call graph for this function:

**13.22.1.2 bml\_threshold\_new()**

```

bml_matrix_t* bml_threshold_new (
    bml_matrix_t * A,
    double threshold )
  
```

Threshold matrix.

**Parameters**

<i>A</i>	Matrix to be thresholded
<i>threshold</i>	Threshold value

**Returns**

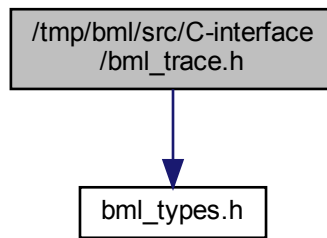
Thresholded A

Here is the call graph for this function:

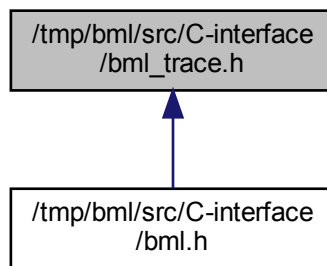
**13.23 /tmp/bml/src/C-interface/bml\_trace.h File Reference**

```
#include "bml_types.h"
```

Include dependency graph for bml\_trace.h:



This graph shows which files directly or indirectly include this file:



## Functions

- double `bml_trace` (`bml_matrix_t *A`)
- double `bml_trace_mult` (`bml_matrix_t *A`, `bml_matrix_t *B`)

### 13.23.1 Function Documentation

#### 13.23.1.1 `bml_trace()`

```
double bml_trace (  
    bml_matrix_t * A )
```

Calculate trace of a matrix.

**Parameters**

<i>A</i>	Matrix to calculate trace for
----------	-------------------------------

**Returns**

Trace of A

Here is the call graph for this function:

**13.23.1.2 bml\_trace\_mult()**

```
double bml_trace_mult (
    bml_matrix_t * A,
    bml_matrix_t * B )
```

Calculate trace of a matrix multiplication.

**Parameters**

<i>A</i>	Matrix A
<i>B</i>	Matrix B

**Returns**

Trace of A\*B

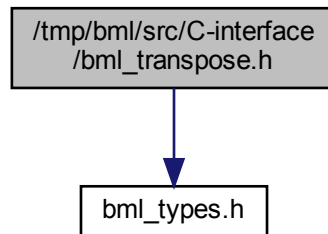
Here is the call graph for this function:



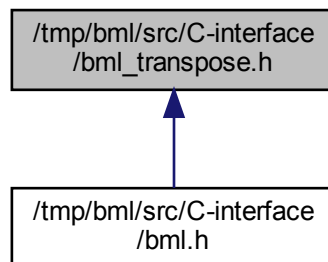


## 13.24 /tmp/bml/src/C-interface/bml\_transpose.h File Reference

```
#include "bml_types.h"
Include dependency graph for bml_transpose.h:
```



This graph shows which files directly or indirectly include this file:



### Functions

- `bml_matrix_t * bml_transpose_new (bml_matrix_t *A)`
- `void bml_transpose (bml_matrix_t *A)`

#### 13.24.1 Function Documentation

##### 13.24.1.1 `bml_transpose()`

```
void bml_transpose (  
    bml_matrix_t * A )
```

Transpose matrix.

## Parameters

<i>A</i>	Matrix to be transposed
----------	-------------------------

## Returns

Transposed A

Here is the call graph for this function:



### 13.24.1.2 bml\_transpose\_new()

```
bml_matrix_t* bml_transpose_new (  
    bml_matrix_t * A )
```

Transpose matrix.

## Parameters

<i>A</i>	Matrix to be transposed
----------	-------------------------

## Returns

Transposed A

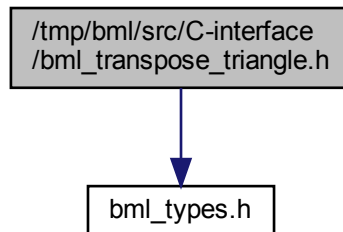
Here is the call graph for this function:



## 13.25 /tmp/bml/src/C-interface/bml\_transpose\_triangle.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_transpose\_triangle.h:



### Functions

- void [bml\\_transpose\\_triangle](#) ([bml\\_matrix\\_t](#) \*A, char triangle)

### 13.25.1 Function Documentation

#### 13.25.1.1 bml\_transpose\_triangle()

```
void bml_transpose_triangle (
    bml\_matrix\_t * A,
    char triangle )
```

Transposes a triangle of a matrix in place.

##### Parameters

<i>A</i>	The matrix for which the triangle should be transposed
<i>triangle</i>	Which triangle to transpose ('u': upper, 'l': lower)

Here is the call graph for this function:



## 13.26 /tmp/bml/src/C-interface/bml\_types.h File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- struct [bml\\_matrix\\_dimension\\_t](#)
- struct [bml\\_domain\\_t](#)

### Typedefs

- typedef void [bml\\_vector\\_t](#)
- typedef void [bml\\_matrix\\_t](#)
- typedef struct [bml\\_domain\\_t](#) [bml\\_domain\\_t](#)

### Enumerations

- enum [bml\\_matrix\\_type\\_t](#) {  
  [type\\_uninitialized](#), [dense](#), [ellpack](#), [ellblock](#),  
  [ellsort](#), [csr](#), [distributed2d](#) }
- enum [bml\\_matrix\\_precision\\_t](#) {  
  [precision\\_uninitialized](#), [single\\_real](#), [double\\_real](#), [single\\_complex](#),  
  [double\\_complex](#) }
- enum [bml\\_dense\\_order\\_t](#) { [dense\\_row\\_major](#), [dense\\_column\\_major](#) }
- enum [bml\\_distribution\\_mode\\_t](#) { [sequential](#), [distributed](#), [graph\\_distributed](#) }

#### 13.26.1 Typedef Documentation

### 13.26.1.1 bml\_matrix\_t

```
typedef void bml_matrix_t
```

The matrix type.

### 13.26.1.2 bml\_vector\_t

```
typedef void bml_vector_t
```

The vector type.

## 13.26.2 Enumeration Type Documentation

### 13.26.2.1 bml\_dense\_order\_t

```
enum bml_dense_order_t
```

The supported dense matrix elements orderings.

Enumerator

dense_row_major	row-major order.
dense_column_major	column-major order.

### 13.26.2.2 bml\_distribution\_mode\_t

```
enum bml_distribution_mode_t
```

The supported distribution modes.

Enumerator

sequential	Each rank works on the full matrix.
distributed	Each rank works on its part of the matrix.
graph_distributed	Each rank works on its set of graph partitions.

### 13.26.2.3 bml\_matrix\_precision\_t

```
enum bml_matrix_precision_t
```

The supported real precisions.

#### Enumerator

precision_uninitialized	The matrix is not initialized.
single_real	Matrix data is stored in single precision (float).
double_real	Matrix data is stored in double precision (double).
single_complex	Matrix data is stored in single-complex precision (float).
double_complex	Matrix data is stored in double-complex precision (double).

#### 13.26.2.4 bml\_matrix\_type\_t

```
enum bml_matrix_type_t
```

The supported matrix types.

#### Enumerator

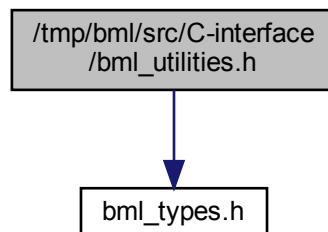
type_uninitialized	The matrix is not initialized.
dense	Dense matrix.
ellpack	ELLPACK matrix.
ellblock	BLOCK ELLPACK matrix.
ellsort	ELLSORT matrix.
csr	CSR matrix.
distributed2d	distributed matrix.

### 13.27 /tmp/bml/src/C-interface/bml\_types\_private.h File Reference

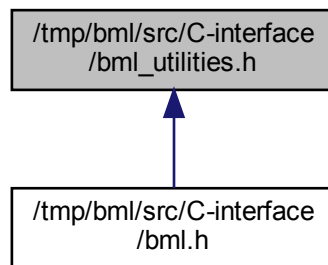
### 13.28 /tmp/bml/src/C-interface/bml\_utilities.h File Reference

```
#include "bml_types.h"
```

Include dependency graph for bml\_utilities.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define PRINT_THRESHOLD 16`

## Functions

- void `bml_print_dense_matrix` (int N, `bml_matrix_precision_t` matrix\_precision, `bml_dense_order_t` order, void \*A, int i\_l, int i\_u, int j\_l, int j\_u)
- void `bml_print_dense_vector` (int N, `bml_matrix_precision_t` matrix\_precision, void \*v, int i\_l, int i\_u)
- void `bml_print_bml_vector` (`bml_vector_t` \*v, int i\_l, int i\_u)
- void `bml_print_bml_matrix` (`bml_matrix_t` \*A, int i\_l, int i\_u, int j\_l, int j\_u)
- void `bml_read_bml_matrix` (`bml_matrix_t` \*A, char \*filename)
- void `bml_write_bml_matrix` (`bml_matrix_t` \*A, char \*filename)
- int `bml_sqrtint` (const int x)

## 13.28.1 Function Documentation

### 13.28.1.1 `bml_print_bml_matrix()`

```

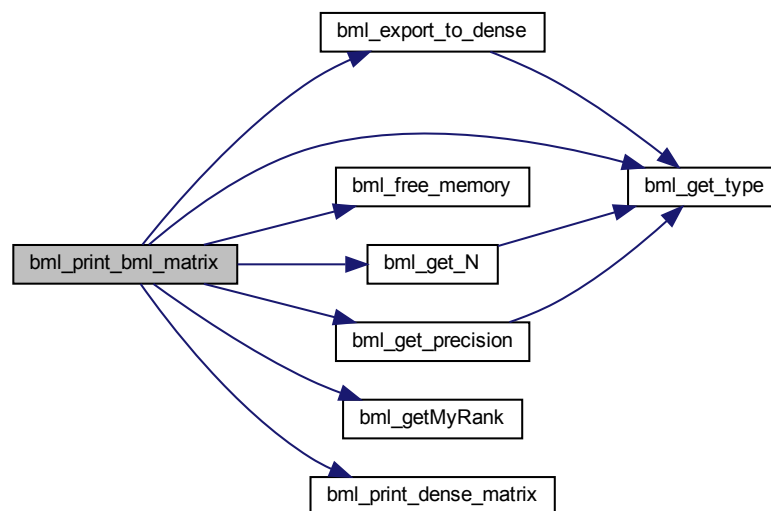
void bml_print_bml_matrix (
    bml_matrix_t * A,
    int i_l,
    int i_u,
    int j_l,
    int j_u )
  
```

Print a dense matrix.

## Parameters

$A$	The matrix.
$i_{\leftarrow}$ $_{\leftarrow}$ $l$	The lower row index.
$i_{\leftarrow}$ $_{\leftarrow}$ $u$	The upper row index.
$j_{\leftarrow}$ $_{\leftarrow}$ $l$	The lower column index.
$j_{\leftarrow}$ $_{\leftarrow}$ $u$	The upper column index.

Here is the call graph for this function:

13.28.1.2 `bml_print_bml_vector()`

```

void bml_print_bml_vector (
    bml_vector_t * v,
    int i_l,
    int i_u )

```

Print a bml vector.

## Parameters

$v$	The vector.
-----	-------------



## Parameters

$i_{\leftarrow}$ $_{\leftarrow}$ $l$	The lower row index.
$i_{\leftarrow}$ $_{\leftarrow}$ $u$	The upper row index.

## 13.28.1.3 bml\_print\_dense\_matrix()

```
void bml_print_dense_matrix (
    int N,
    bml_matrix_precision_t matrix_precision,
    bml_dense_order_t order,
    void * A,
    int i_l,
    int i_u,
    int j_l,
    int j_u )
```

Print a dense matrix.

## Parameters

$N$	The number of rows/columns.
$matrix\_precision$	The real precision.
$order$	The matrix element order.
$A$	The matrix.
$i\_l$	The lower row index.
$i\_u$	The upper row index.
$j\_l$	The lower column index.
$j\_u$	The upper column index.

Here is the caller graph for this function:



### 13.28.1.4 bml\_print\_dense\_vector()

```
void bml_print_dense_vector (
    int N,
    bml_matrix_precision_t matrix_precision,
    void * v,
    int i_l,
    int i_u )
```

Print a dense vector.

#### Parameters

<i>N</i>	The number of rows/columns.
<i>matrix_precision</i>	The real precision.
<i>v</i>	The vector.
<i>i_l</i>	The lower row index.
<i>i_u</i>	The upper row index.

### 13.28.1.5 bml\_read\_bml\_matrix()

```
void bml_read_bml_matrix (
    bml_matrix_t * A,
    char * filename )
```

Read a bml matrix from a Matrix Market file.

#### Parameters

<i>A</i>	The matrix
<i>filename</i>	The file containing matrix

Here is the call graph for this function:



### 13.28.1.6 bml\_write\_bml\_matrix()

```
void bml_write_bml_matrix (
```

```
bml_matrix_t * A,  
char * filename )
```

Write a bml matrix to a Matrix Market file.

#### Parameters

<i>A</i>	The matrix
<i>filename</i>	The file containing matrix

Here is the call graph for this function:





# Index

/tmp/bml/src/C-interface/bml.h, [47](#)  
/tmp/bml/src/C-interface/bml\_add.h, [48](#)  
/tmp/bml/src/C-interface/bml\_adjungate\_triangle.h, [48](#)  
/tmp/bml/src/C-interface/bml\_allocate.h, [50](#)  
/tmp/bml/src/C-interface/bml\_convert.h, [51](#)  
/tmp/bml/src/C-interface/bml\_copy.h, [52](#)  
/tmp/bml/src/C-interface/bml\_element\_multiply.h, [56](#)  
/tmp/bml/src/C-interface/bml\_export.h, [58](#)  
/tmp/bml/src/C-interface/bml\_getters.h, [59](#)  
/tmp/bml/src/C-interface/bml\_import.h, [62](#)  
/tmp/bml/src/C-interface/bml\_init.h, [63](#)  
/tmp/bml/src/C-interface/bml\_introspection.h, [65](#)  
/tmp/bml/src/C-interface/bml\_logger.h, [72](#)  
/tmp/bml/src/C-interface/bml\_multiply.h, [75](#)  
/tmp/bml/src/C-interface/bml\_norm.h, [79](#)  
/tmp/bml/src/C-interface/bml\_normalize.h, [84](#)  
/tmp/bml/src/C-interface/bml\_parallel.h, [87](#)  
/tmp/bml/src/C-interface/bml\_scale.h, [89](#)  
/tmp/bml/src/C-interface/bml\_setters.h, [92](#)  
/tmp/bml/src/C-interface/bml\_shutdown.h, [93](#)  
/tmp/bml/src/C-interface/bml\_submatrix.h, [95](#)  
/tmp/bml/src/C-interface/bml\_threshold.h, [100](#)  
/tmp/bml/src/C-interface/bml\_trace.h, [102](#)  
/tmp/bml/src/C-interface/bml\_transpose.h, [105](#)  
/tmp/bml/src/C-interface/bml\_transpose\_triangle.h, [107](#)  
/tmp/bml/src/C-interface/bml\_types.h, [108](#)  
/tmp/bml/src/C-interface/bml\_types\_private.h, [110](#)  
/tmp/bml/src/C-interface/bml\_utilities.h, [110](#)

## Add Functions (C interface), [35](#)

bml\_add, [35](#)  
bml\_add\_identity, [35](#)  
bml\_add\_norm, [36](#)  
bml\_scale\_add\_identity, [37](#)

## Add Functions (Fortran interface), [41](#)

## Allocation and Deallocation Functions (C interface), [23](#)

bml\_allocate\_memory, [23](#)  
bml\_allocated, [24](#)  
bml\_banded\_matrix, [24](#)  
bml\_clear, [25](#)  
bml\_deallocate, [25](#)  
bml\_deallocate\_domain, [26](#)  
bml\_default\_domain, [26](#)  
bml\_free\_memory, [27](#)  
bml\_free\_ptr, [28](#)  
bml\_identity\_matrix, [28](#)  
bml\_noinit\_allocate\_memory, [29](#)  
bml\_noinit\_matrix, [29](#)  
bml\_noinit\_rectangular\_matrix, [30](#)  
bml\_random\_matrix, [31](#)

bml\_reallocate\_memory, [31](#)

bml\_update\_domain, [32](#)

bml\_zero\_matrix, [32](#)

## Allocation and Deallocation Functions (Fortran interface), [40](#)

## bml\_add

Add Functions (C interface), [35](#)

## bml\_add\_identity

Add Functions (C interface), [35](#)

## bml\_add\_norm

Add Functions (C interface), [36](#)

## bml\_adjacency

bml\_submatrix.h, [96](#)

## bml\_adjacency\_group

bml\_submatrix.h, [96](#)

## bml\_adjungate\_triangle

bml\_adjungate\_triangle.h, [49](#)

## bml\_adjungate\_triangle.h

bml\_adjungate\_triangle, [49](#)

## bml\_allGatherVParallel

bml\_parallel.h, [88](#)

## bml\_allocate\_memory

Allocation and Deallocation Functions (C interface),  
[23](#)

## bml\_allocated

Allocation and Deallocation Functions (C interface),  
[24](#)

## bml\_banded\_matrix

Allocation and Deallocation Functions (C interface),  
[24](#)

## bml\_clear

Allocation and Deallocation Functions (C interface),  
[25](#)

## bml\_convert

bml\_convert.h, [52](#)

## bml\_convert.h

bml\_convert, [52](#)

## bml\_copy

bml\_copy.h, [53](#)

## bml\_copy.h

bml\_copy, [53](#)

bml\_copy\_domain, [54](#)

bml\_copy\_new, [54](#)

bml\_reorder, [55](#)

bml\_restore\_domain, [55](#)

bml\_save\_domain, [56](#)

## bml\_copy\_domain

bml\_copy.h, [54](#)

## bml\_copy\_new

- bml\_copy.h, 54
- bml\_deallocate
  - Allocation and Deallocation Functions (C interface), 25
- bml\_deallocate\_domain
  - Allocation and Deallocation Functions (C interface), 26
- bml\_default\_domain
  - Allocation and Deallocation Functions (C interface), 26
- bml\_dense\_order\_t
  - bml\_types.h, 109
- bml\_distribution\_mode\_t
  - bml\_types.h, 109
- bml\_domain\_t, 43
  - globalRowExtent, 43
  - globalRowMax, 43
  - globalRowMin, 44
  - localDispl, 44
  - localElements, 44
  - localRowExtent, 44
  - localRowMax, 44
  - localRowMin, 44
  - maxLocalExtent, 44
  - minLocalExtent, 44
  - totalCols, 45
  - totalProcs, 45
  - totalRows, 45
- bml\_element\_multiply.h
  - bml\_element\_multiply\_AB, 57
- bml\_element\_multiply\_AB
  - bml\_element\_multiply.h, 57
- bml\_export\_to\_dense
  - Converting between Matrix Formats (C interface), 38
- bml\_fnorm
  - bml\_norm.h, 80
- bml\_fnorm2
  - bml\_norm.h, 81
- bml\_free\_memory
  - Allocation and Deallocation Functions (C interface), 27
- bml\_free\_ptr
  - Allocation and Deallocation Functions (C interface), 28
- bml\_gershgorin
  - bml\_normalize.h, 85
- bml\_gershgorin\_partial
  - bml\_normalize.h, 86
- bml\_get\_bandwidth
  - bml\_introspection.h, 66
- bml\_get\_deep\_type
  - bml\_introspection.h, 66
- bml\_get\_diagonal
  - bml\_getters.h, 60
- bml\_get\_distribution\_mode
  - bml\_introspection.h, 67
- bml\_get\_element
  - bml\_getters.h, 61
- bml\_get\_M
  - bml\_introspection.h, 67
- bml\_get\_N
  - bml\_introspection.h, 68
- bml\_get\_precision
  - bml\_introspection.h, 69
- bml\_get\_row
  - bml\_getters.h, 61
- bml\_get\_row\_bandwidth
  - bml\_introspection.h, 70
- bml\_get\_sparsity
  - bml\_introspection.h, 71
- bml\_get\_type
  - bml\_introspection.h, 71
- bml\_getMyRank
  - bml\_parallel.h, 88
- bml\_getNRanks
  - bml\_parallel.h, 89
- bml\_getters.h
  - bml\_get\_diagonal, 60
  - bml\_get\_element, 61
  - bml\_get\_row, 61
- bml\_group\_matrix
  - bml\_submatrix.h, 97
- bml\_identity\_matrix
  - Allocation and Deallocation Functions (C interface), 28
- bml\_import\_from\_dense
  - Converting between Matrix Formats (C interface), 39
- bml\_init
  - bml\_init.h, 64
- bml\_init.h
  - bml\_init, 64
  - bml\_initF, 64
- bml\_initF
  - bml\_init.h, 64
- bml\_introspection.h
  - bml\_get\_bandwidth, 66
  - bml\_get\_deep\_type, 66
  - bml\_get\_distribution\_mode, 67
  - bml\_get\_M, 67
  - bml\_get\_N, 68
  - bml\_get\_precision, 69
  - bml\_get\_row\_bandwidth, 70
  - bml\_get\_sparsity, 71
  - bml\_get\_type, 71
- bml\_log
  - bml\_logger.h, 74
- BML\_LOG\_DEBUG
  - bml\_logger.h, 74
- BML\_LOG\_ERROR
  - bml\_logger.h, 74
- BML\_LOG\_INFO
  - bml\_logger.h, 74
- bml\_log\_level\_t
  - bml\_logger.h, 74

- bml\_log\_location
  - bml\_logger.h, 75
- BML\_LOG\_WARNING
  - bml\_logger.h, 74
- bml\_logger.h
  - bml\_log, 74
  - BML\_LOG\_DEBUG, 74
  - BML\_LOG\_ERROR, 74
  - BML\_LOG\_INFO, 74
  - bml\_log\_level\_t, 74
  - bml\_log\_location, 75
  - BML\_LOG\_WARNING, 74
  - LOG\_DEBUG, 73
  - LOG\_ERROR, 73
  - LOG\_INFO, 74
  - LOG\_WARN, 74
- bml\_matrix2submatrix
  - bml\_submatrix.h, 97
- bml\_matrix2submatrix\_index
  - bml\_submatrix.h, 98
- bml\_matrix2submatrix\_index\_graph
  - bml\_submatrix.h, 99
- bml\_matrix\_dimension\_t, 45
  - bsizes, 46
  - N\_cols, 46
  - N\_nz\_max, 46
  - N\_rows, 46
  - NB, 46
- bml\_matrix\_precision\_t
  - bml\_types.h, 109
- bml\_matrix\_t
  - bml\_types.h, 108
- bml\_matrix\_type\_t
  - bml\_types.h, 110
- bml\_multiply
  - bml\_multiply.h, 76
- bml\_multiply.h
  - bml\_multiply, 76
  - bml\_multiply\_AB, 77
  - bml\_multiply\_adjust\_AB, 78
  - bml\_multiply\_x2, 78
- bml\_multiply\_AB
  - bml\_multiply.h, 77
- bml\_multiply\_adjust\_AB
  - bml\_multiply.h, 78
- bml\_multiply\_x2
  - bml\_multiply.h, 78
- bml\_noinit\_allocate\_memory
  - Allocation and Deallocation Functions (C interface), 29
- bml\_noinit\_matrix
  - Allocation and Deallocation Functions (C interface), 29
- bml\_noinit\_rectangular\_matrix
  - Allocation and Deallocation Functions (C interface), 30
- bml\_norm.h
  - bml\_fnorm, 80
  - bml\_fnorm2, 81
  - bml\_sum\_AB, 81
  - bml\_sum\_squares, 82
  - bml\_sum\_squares2, 83
  - bml\_sum\_squares\_submatrix, 83
- bml\_normalize
  - bml\_normalize.h, 86
- bml\_normalize.h
  - bml\_gershgorin, 85
  - bml\_gershgorin\_partial, 86
  - bml\_normalize, 86
- bml\_parallel.h
  - bml\_allGatherVParallel, 88
  - bml\_getMyRank, 88
  - bml\_getNRanks, 89
- bml\_print\_bml\_matrix
  - bml\_utilities.h, 111
- bml\_print\_bml\_vector
  - bml\_utilities.h, 112
- bml\_print\_dense\_matrix
  - bml\_utilities.h, 113
- bml\_print\_dense\_vector
  - bml\_utilities.h, 113
- bml\_random\_matrix
  - Allocation and Deallocation Functions (C interface), 31
- bml\_read\_bml\_matrix
  - bml\_utilities.h, 114
- bml\_reallocate\_memory
  - Allocation and Deallocation Functions (C interface), 31
- bml\_reorder
  - bml\_copy.h, 55
- bml\_restore\_domain
  - bml\_copy.h, 55
- bml\_save\_domain
  - bml\_copy.h, 56
- bml\_scale
  - bml\_scale.h, 90
- bml\_scale.h
  - bml\_scale, 90
  - bml\_scale\_inplace, 91
  - bml\_scale\_new, 91
- bml\_scale\_add\_identity
  - Add Functions (C interface), 37
- bml\_scale\_inplace
  - bml\_scale.h, 91
- bml\_scale\_new
  - bml\_scale.h, 91
- bml\_shutdown
  - bml\_shutdown.h, 94
- bml\_shutdown.h
  - bml\_shutdown, 94
  - bml\_shutdownF, 94
- bml\_shutdownF
  - bml\_shutdown.h, 94
- bml\_submatrix.h
  - bml\_adjacency, 96

- bml\_adjacency\_group, 96
  - bml\_group\_matrix, 97
  - bml\_matrix2submatrix, 97
  - bml\_matrix2submatrix\_index, 98
  - bml\_matrix2submatrix\_index\_graph, 99
  - bml\_submatrix2matrix, 99
- bml\_submatrix2matrix
  - bml\_submatrix.h, 99
- bml\_sum\_AB
  - bml\_norm.h, 81
- bml\_sum\_squares
  - bml\_norm.h, 82
- bml\_sum\_squares2
  - bml\_norm.h, 83
- bml\_sum\_squares\_submatrix
  - bml\_norm.h, 83
- bml\_threshold
  - bml\_threshold.h, 101
- bml\_threshold.h
  - bml\_threshold, 101
  - bml\_threshold\_new, 102
- bml\_threshold\_new
  - bml\_threshold.h, 102
- bml\_trace
  - bml\_trace.h, 103
- bml\_trace.h
  - bml\_trace, 103
  - bml\_trace\_mult, 104
- bml\_trace\_mult
  - bml\_trace.h, 104
- bml\_transpose
  - bml\_transpose.h, 105
- bml\_transpose.h
  - bml\_transpose, 105
  - bml\_transpose\_new, 106
- bml\_transpose\_new
  - bml\_transpose.h, 106
- bml\_transpose\_triangle
  - bml\_transpose\_triangle.h, 107
- bml\_transpose\_triangle.h
  - bml\_transpose\_triangle, 107
- bml\_types.h
  - bml\_dense\_order\_t, 109
  - bml\_distribution\_mode\_t, 109
  - bml\_matrix\_precision\_t, 109
  - bml\_matrix\_t, 108
  - bml\_matrix\_type\_t, 110
  - bml\_vector\_t, 109
  - csr, 110
  - dense, 110
  - dense\_column\_major, 109
  - dense\_row\_major, 109
  - distributed, 109
  - distributed2d, 110
  - double\_complex, 110
  - double\_real, 110
  - ellblock, 110
  - ellpack, 110
- ellsort, 110
  - graph\_distributed, 109
  - precision\_uninitialized, 110
  - sequential, 109
  - single\_complex, 110
  - single\_real, 110
  - type\_uninitialized, 110
- bml\_update\_domain
  - Allocation and Deallocation Functions (C interface), 32
- bml\_utilities.h
  - bml\_print\_bml\_matrix, 111
  - bml\_print\_bml\_vector, 112
  - bml\_print\_dense\_matrix, 113
  - bml\_print\_dense\_vector, 113
  - bml\_read\_bml\_matrix, 114
  - bml\_write\_bml\_matrix, 114
- bml\_vector\_t
  - bml\_types.h, 109
- bml\_write\_bml\_matrix
  - bml\_utilities.h, 114
- bml\_zero\_matrix
  - Allocation and Deallocation Functions (C interface), 32
- bsizes
  - bml\_matrix\_dimension\_t, 46
- Converting between Matrix Formats (C interface), 38
  - bml\_export\_to\_dense, 38
  - bml\_import\_from\_dense, 39
- Converting between Matrix Formats (Fortran interface), 42
- csr
  - bml\_types.h, 110
- dense
  - bml\_types.h, 110
- dense\_column\_major
  - bml\_types.h, 109
- dense\_row\_major
  - bml\_types.h, 109
- distributed
  - bml\_types.h, 109
- distributed2d
  - bml\_types.h, 110
- double\_complex
  - bml\_types.h, 110
- double\_real
  - bml\_types.h, 110
- ellblock
  - bml\_types.h, 110
- ellpack
  - bml\_types.h, 110
- ellsort
  - bml\_types.h, 110
- globalRowExtent
  - bml\_domain\_t, 43



- globalRowMax
  - bml\_domain\_t, [43](#)
- globalRowMin
  - bml\_domain\_t, [44](#)
- graph\_distributed
  - bml\_types.h, [109](#)
- localDispl
  - bml\_domain\_t, [44](#)
- localElements
  - bml\_domain\_t, [44](#)
- localRowExtent
  - bml\_domain\_t, [44](#)
- localRowMax
  - bml\_domain\_t, [44](#)
- localRowMin
  - bml\_domain\_t, [44](#)
- LOG\_DEBUG
  - bml\_logger.h, [73](#)
- LOG\_ERROR
  - bml\_logger.h, [73](#)
- LOG\_INFO
  - bml\_logger.h, [74](#)
- LOG\_WARN
  - bml\_logger.h, [74](#)
- maxLocalExtent
  - bml\_domain\_t, [44](#)
- minLocalExtent
  - bml\_domain\_t, [44](#)
- N\_cols
  - bml\_matrix\_dimension\_t, [46](#)
- N\_nz\_max
  - bml\_matrix\_dimension\_t, [46](#)
- N\_rows
  - bml\_matrix\_dimension\_t, [46](#)
- NB
  - bml\_matrix\_dimension\_t, [46](#)
- precision\_uninitialized
  - bml\_types.h, [110](#)
- sequential
  - bml\_types.h, [109](#)
- single\_complex
  - bml\_types.h, [110](#)
- single\_real
  - bml\_types.h, [110](#)
- totalCols
  - bml\_domain\_t, [45](#)
- totalProcs
  - bml\_domain\_t, [45](#)
- totalRows
  - bml\_domain\_t, [45](#)
- type\_uninitialized
  - bml\_types.h, [110](#)