

bml

2.4.0

Generated by Doxygen 1.8.17

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

11.1.2.9 bml_free_ptr()	26
11.1.2.10 bml_identity_matrix()	27
11.1.2.11 bml_noinit_allocate_memory()	27
11.1.2.12 bml_noinit_matrix()	27
11.1.2.13 bml_noinit_rectangular_matrix()	28
11.1.2.14 bml_random_matrix()	28
11.1.2.15 bml_reallocate_memory()	29
11.1.2.16 bml_update_domain_matrix()	29
11.1.2.17 bml_zero_matrix()	30
11.2 Add Functions (C interface)	31
11.2.1 Detailed Description	31
11.2.2 Function Documentation	31
11.2.2.1 bml_add()	31
11.2.2.2 bml_add_identity()	31
11.2.2.3 bml_add_norm()	32
11.2.2.4 bml_scale_add_identity()	32
11.3 Converting between Matrix Formats (C interface)	33
11.3.1 Detailed Description	33
11.3.2 Function Documentation	33
11.3.2.1 bml_export_to_dense()	33
11.3.2.2 bml_import_from_dense()	33
11.4 Allocation and Deallocation Functions (Fortran interface)	35
11.5 Add Functions (Fortran interface)	36
11.6 Converting between Matrix Formats (Fortran interface)	37
12 Class Documentation	39
12.1 bml_domain_t Struct Reference	39
12.1.1 Detailed Description	39
12.1.2 Member Data Documentation	39
12.1.2.1 globalRowExtent	39
12.1.2.2 globalRowMax	40
12.1.2.3 globalRowMin	40
12.1.2.4 localDispl	40
12.1.2.5 localElements	40
12.1.2.6 localRowExtent	40
12.1.2.7 localRowMax	40
12.1.2.8 localRowMin	40
12.1.2.9 maxLocalExtent	40
12.1.2.10 minLocalExtent	41
12.1.2.11 totalCols	41
12.1.2.12 totalProcs	41
12.1.2.13 totalRows	41

12.2 bml_matrix_dimension_t Struct Reference	41
12.2.1 Detailed Description	41
12.2.2 Member Data Documentation	42
12.2.2.1 bsizes	42
12.2.2.2 N_cols	42
12.2.2.3 N_nz_max	42
12.2.2.4 N_rows	42
12.2.2.5 NB	42
13 File Documentation	43
13.1 /bml/src/C-interface/bml.h File Reference	43
13.1.1 Detailed Description	43
13.2 /bml/src/C-interface/bml_add.h File Reference	44
13.3 /bml/src/C-interface/bml_adjungate_triangle.h File Reference	44
13.3.1 Function Documentation	44
13.3.1.1 bml_adjungate_triangle()	44
13.4 /bml/src/C-interface/bml_allocate.h File Reference	44
13.5 /bml/src/C-interface/bml_convert.h File Reference	45
13.5.1 Function Documentation	45
13.5.1.1 bml_convert()	45
13.6 /bml/src/C-interface/bml_copy.h File Reference	46
13.6.1 Function Documentation	46
13.6.1.1 bml_copy()	46
13.6.1.2 bml_copy_new()	46
13.6.1.3 bml_reorder()	47
13.6.1.4 bml_restore_domain()	47
13.6.1.5 bml_save_domain()	47
13.7 /bml/src/C-interface/bml_element_multiply.h File Reference	48
13.7.1 Function Documentation	48
13.7.1.1 bml_element_multiply_AB()	48
13.8 /bml/src/C-interface/bml_export.h File Reference	48
13.9 /bml/src/C-interface/bml_getters.h File Reference	48
13.9.1 Function Documentation	49
13.9.1.1 bml_get_diagonal()	49
13.9.1.2 bml_get_element()	49
13.9.1.3 bml_get_row()	50
13.10 /bml/src/C-interface/bml_import.h File Reference	50
13.11 /bml/src/C-interface/bml_init.h File Reference	50
13.11.1 Function Documentation	50
13.11.1.1 bml_init()	50
13.11.1.2 bml_initF()	51
13.12 /bml/src/C-interface/bml_introspection.h File Reference	51

13.12.1 Function Documentation	51
13.12.1.1 bml_get_bandwidth()	51
13.12.1.2 bml_get_deep_type()	52
13.12.1.3 bml_get_distribution_mode()	52
13.12.1.4 bml_get_M()	52
13.12.1.5 bml_get_N()	54
13.12.1.6 bml_get_precision()	54
13.12.1.7 bml_get_row_bandwidth()	54
13.12.1.8 bml_get_sparsity()	55
13.12.1.9 bml_get_type()	55
13.13 /bml/src/C-interface/bml_logger.h File Reference	56
13.13.1 Macro Definition Documentation	56
13.13.1.1 LOG_DEBUG	56
13.13.1.2 LOG_ERROR	56
13.13.1.3 LOG_INFO	57
13.13.1.4 LOG_WARN	57
13.13.2 Enumeration Type Documentation	57
13.13.2.1 bml_log_level_t	57
13.13.3 Function Documentation	57
13.13.3.1 bml_log()	57
13.13.3.2 bml_log_location()	58
13.14 /bml/src/C-interface/bml_multiply.h File Reference	58
13.14.1 Function Documentation	58
13.14.1.1 bml_multiply()	59
13.14.1.2 bml_multiply_AB()	59
13.14.1.3 bml_multiply_adjust_AB()	59
13.14.1.4 bml_multiply_x2()	60
13.15 /bml/src/C-interface/bml_norm.h File Reference	60
13.15.1 Function Documentation	61
13.15.1.1 bml_fnorm()	61
13.15.1.2 bml_fnorm2()	61
13.15.1.3 bml_sum_AB()	61
13.15.1.4 bml_sum_squares()	62
13.15.1.5 bml_sum_squares2()	62
13.15.1.6 bml_sum_squares_submatrix()	63
13.16 /bml/src/C-interface/bml_normalize.h File Reference	63
13.16.1 Function Documentation	63
13.16.1.1 bml_gershgorin()	63
13.16.1.2 bml_gershgorin_partial()	64
13.16.1.3 bml_normalize()	64
13.17 /bml/src/C-interface/bml_parallel.h File Reference	64
13.17.1 Function Documentation	65

13.17.1.1 bml_allGatherVParallel()	65
13.17.1.2 bml_getMyRank()	65
13.17.1.3 bml_getNRanks()	65
13.18 /bml/src/C-interface/bml_scale.h File Reference	66
13.18.1 Function Documentation	66
13.18.1.1 bml_scale()	66
13.18.1.2 bml_scale_inplace()	66
13.18.1.3 bml_scale_new()	67
13.19 /bml/src/C-interface/bml_setters.h File Reference	67
13.20 /bml/src/C-interface/bml_shutdown.h File Reference	67
13.20.1 Function Documentation	67
13.20.1.1 bml_shutdown()	68
13.20.1.2 bml_shutdownF()	68
13.21 /bml/src/C-interface/bml_submatrix.h File Reference	68
13.21.1 Function Documentation	68
13.21.1.1 bml_adjacency()	68
13.21.1.2 bml_adjacency_group()	69
13.21.1.3 bml_group_matrix()	69
13.21.1.4 bml_matrix2submatrix()	70
13.21.1.5 bml_matrix2submatrix_index()	70
13.21.1.6 bml_matrix2submatrix_index_graph()	70
13.21.1.7 bml_submatrix2matrix()	71
13.22 /bml/src/C-interface/bml_threshold.h File Reference	71
13.22.1 Function Documentation	72
13.22.1.1 bml_threshold()	72
13.22.1.2 bml_threshold_new()	72
13.23 /bml/src/C-interface/bml_trace.h File Reference	72
13.23.1 Function Documentation	73
13.23.1.1 bml_trace()	73
13.23.1.2 bml_trace_mult()	73
13.24 /bml/src/C-interface/bml_transpose.h File Reference	73
13.24.1 Function Documentation	74
13.24.1.1 bml_transpose()	74
13.24.1.2 bml_transpose_new()	74
13.25 /bml/src/C-interface/bml_transpose_triangle.h File Reference	74
13.25.1 Function Documentation	75
13.25.1.1 bml_transpose_triangle()	75
13.26 /bml/src/C-interface/bml_types.h File Reference	75
13.26.1 Typedef Documentation	75
13.26.1.1 bml_matrix_t	76
13.26.1.2 bml_vector_t	76
13.26.2 Enumeration Type Documentation	76

13.26.2.1 bml_dense_order_t	76
13.26.2.2 bml_distribution_mode_t	76
13.26.2.3 bml_matrix_precision_t	76
13.26.2.4 bml_matrix_type_t	77
13.27 /bml/src/C-interface/bml_types_private.h File Reference	77
13.28 /bml/src/C-interface/bml_utilities.h File Reference	77
13.28.1 Function Documentation	78
13.28.1.1 bml_print_bml_matrix()	78
13.28.1.2 bml_print_bml_vector()	78
13.28.1.3 bml_print_dense_matrix()	79
13.28.1.4 bml_print_dense_vector()	79
13.28.1.5 bml_read_bml_matrix()	80
13.28.1.6 bml_write_bml_matrix()	80
Index	81

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)

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, 100×100 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)	23
Add Functions (C interface)	31
Converting between Matrix Formats (C interface)	33
Allocation and Deallocation Functions (Fortran interface)	35
Add Functions (Fortran interface)	36
Converting between Matrix Formats (Fortran interface)	37

Chapter 9

Class Index

9.1 Class List

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

bml_domain_t	39
bml_matrix_dimension_t	41

Chapter 10

File Index

10.1 File List

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

/bml/src/C-interface/ blas.h	??
/bml/src/C-interface/ bml.h	43
/bml/src/C-interface/ bml_add.h	44
/bml/src/C-interface/ bml_adjungate_triangle.h	44
/bml/src/C-interface/ bml_allocate.h	44
/bml/src/C-interface/ bml_convert.h	45
/bml/src/C-interface/ bml_copy.h	46
/bml/src/C-interface/ bml_diagonalize.h	??
/bml/src/C-interface/ bml_domain.h	??
/bml/src/C-interface/ bml_element_multiply.h	48
/bml/src/C-interface/ bml_elemental.h	??
/bml/src/C-interface/ bml_export.h	48
/bml/src/C-interface/ bml_getters.h	48
/bml/src/C-interface/ bml_import.h	50
/bml/src/C-interface/ bml_init.h	50
/bml/src/C-interface/ bml_introspection.h	51
/bml/src/C-interface/ bml_inverse.h	??
/bml/src/C-interface/ bml_logger.h	56
/bml/src/C-interface/ bml_multiply.h	58
/bml/src/C-interface/ bml_norm.h	60
/bml/src/C-interface/ bml_normalize.h	63
/bml/src/C-interface/ bml_parallel.h	64
/bml/src/C-interface/ bml_scale.h	66
/bml/src/C-interface/ bml_setters.h	67
/bml/src/C-interface/ bml_shutdown.h	67
/bml/src/C-interface/ bml_submatrix.h	68
/bml/src/C-interface/ bml_threshold.h	71
/bml/src/C-interface/ bml_trace.h	72
/bml/src/C-interface/ bml_transpose.h	73
/bml/src/C-interface/ bml_transpose_triangle.h	74
/bml/src/C-interface/ bml_types.h	75
/bml/src/C-interface/ bml_types_private.h	77
/bml/src/C-interface/ bml_utilities.h	77
/bml/src/C-interface/ lapack.h	??

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_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)`
- `void bml_update_domain_matrix (bml_matrix_t *A, int *localPartMin, int *localPartMax, int *nnodesInPart)`
- `bml_domain_t * bml_default_domain (int N, int M, bml_distribution_mode_t distrib_mode)`
- `void bml_deallocate_domain (bml_domain_t *D)`

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.

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

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.
------------------	-------------

11.1.2.5 bml_deallocate()

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

Deallocate a matrix.

Parameters

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

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.
------------------	-------------

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 rank

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.
------------	--

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.
------------	--

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.

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.

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_matrix()

```

void bml_update_domain_matrix (
    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

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

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

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

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.

$$A \leftarrow \alpha A + \beta Id$$

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

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

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:

- [/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:

- [/bml/src/C-interface/bml_types.h](#)

Chapter 13

File Documentation

13.1 /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"
```

13.1.1 Detailed Description

Copyright

Los Alamos National Laboratory 2015

13.2 /bml/src/C-interface/bml_add.h File Reference

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

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 /bml/src/C-interface/bml_adjungate_triangle.h File Reference

```
#include "bml_types.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)

13.4 /bml/src/C-interface/bml_allocate.h File Reference

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

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_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)
- void `bml_update_domain_matrix` (`bml_matrix_t` *A, int *localPartMin, int *localPartMax, int *nnodesInPart)

13.5 /bml/src/C-interface/bml_convert.h File Reference

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

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

A	The input matrix.
-----	-------------------

Returns

The converted matrix B .

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

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

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_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

A	Matrix to copy
B	Copy of Matrix A

13.6.1.2 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

13.6.1.3 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

13.6.1.4 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
----------	--------------------

13.6.1.5 bml_save_domain()

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

Save current domain for bml matrix.

Parameters

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

13.7 /bml/src/C-interface/bml_element_multiply.h File Reference

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

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

13.8 /bml/src/C-interface/bml_export.h File Reference

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

Functions

- void * [bml_export_to_dense](#) ([bml_matrix_t](#) *A, [bml_dense_order_t](#) order)

13.9 /bml/src/C-interface/bml_getters.h File Reference

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


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)

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

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).

13.10 /bml/src/C-interface/bml_import.h File Reference

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

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 /bml/src/C-interface/bml_init.h File Reference

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

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 /bml/src/C-interface/bml_introspection.h File Reference

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

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*.

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

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 distibution mode of matrix *A*.

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.

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.

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.

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*.

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*.

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 /bml/src/C-interface/bml_logger.h File Reference

```
#include "bml_types.h"
#include <stdlib.h>
#include <stdio.h>
```

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 linenummer, char *format,...)
- char * [bml_version](#) (void)
Return version string of library.
- void [bml_print_version](#) (void)

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 /bml/src/C-interface/bml_multiply.h File Reference

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

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

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

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

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

13.15 /bml/src/C-interface/bml_norm.h File Reference

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

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

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

Returns

Frobenius norm of Matrix A

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

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 $\alpha * A + \beta * B$

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

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 $\alpha * A + \beta * B$

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

13.16 /bml/src/C-interface/bml_normalize.h File Reference

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

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

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

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

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

13.17 /bml/src/C-interface/bml_parallel.h File Reference

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

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
----------	----------

13.17.1.2 bml_getMyRank()

```
int bml_getMyRank (
    void )
```

Get local MPI rank.

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.

13.18 /bml/src/C-interface/bml_scale.h File Reference

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

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

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

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
A	Matrix to scale

Returns

A Scaled Copy of A

13.19 /bml/src/C-interface/bml_setters.h File Reference

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

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 /bml/src/C-interface/bml_shutdown.h File Reference

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

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 /bml/src/C-interface/bml_submatrix.h File Reference

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

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

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
<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

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

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

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)

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)

13.21.1.7 bml_submatrix2matrix()

```

void bml_submatrix2matrix (
    bml_matrix_t * A,
    bml_matrix_t * B,
    int * core_halo_index,
    int lsize,
    int lsize,
    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>lsize</i>	Number of core positions

13.22 /bml/src/C-interface/bml_threshold.h File Reference

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

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

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

13.23 `/bml/src/C-interface/bml_trace.h` File Reference

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

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

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

13.24 /bml/src/C-interface/bml_transpose.h File Reference

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

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

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

Returns

Transposed A

13.24.1.2 `bml_transpose_new()`

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

Transpose matrix.

Parameters

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

Returns

Transposed A

13.25 `/bml/src/C-interface/bml_transpose_triangle.h` File Reference

```
#include "bml_types.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)

13.26 /bml/src/C-interface/bml_types.h File Reference

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](#),
 [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.
csr	CSR matrix.
distributed2d	distributed matrix.

13.27 /bml/src/C-interface/bml_types_private.h File Reference

13.28 /bml/src/C-interface/bml_utilities.h File Reference

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

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

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

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

<i>v</i>	The vector.
i_{\leftarrow} $_{\leftarrow}$ <i>l</i>	The lower row index.
i_{\leftarrow} $_{\leftarrow}$ <i>u</i>	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

<i>N</i>	The number of rows/columns.
<i>matrix_precision</i>	The real precision.
<i>order</i>	The matrix element order.
<i>A</i>	The matrix.
<i>i_l</i>	The lower row index.
<i>i_u</i>	The upper row index.
<i>j_l</i>	The lower column index.
<i>j_u</i>	The upper column index.

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

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

Index

/bml/src/C-interface/bml.h, [43](#)
/bml/src/C-interface/bml_add.h, [44](#)
/bml/src/C-interface/bml_adjungate_triangle.h, [44](#)
/bml/src/C-interface/bml_allocate.h, [44](#)
/bml/src/C-interface/bml_convert.h, [45](#)
/bml/src/C-interface/bml_copy.h, [46](#)
/bml/src/C-interface/bml_element_multiply.h, [48](#)
/bml/src/C-interface/bml_export.h, [48](#)
/bml/src/C-interface/bml_getters.h, [48](#)
/bml/src/C-interface/bml_import.h, [50](#)
/bml/src/C-interface/bml_init.h, [50](#)
/bml/src/C-interface/bml_introspection.h, [51](#)
/bml/src/C-interface/bml_logger.h, [56](#)
/bml/src/C-interface/bml_multiply.h, [58](#)
/bml/src/C-interface/bml_norm.h, [60](#)
/bml/src/C-interface/bml_normalize.h, [63](#)
/bml/src/C-interface/bml_parallel.h, [64](#)
/bml/src/C-interface/bml_scale.h, [66](#)
/bml/src/C-interface/bml_setters.h, [67](#)
/bml/src/C-interface/bml_shutdown.h, [67](#)
/bml/src/C-interface/bml_submatrix.h, [68](#)
/bml/src/C-interface/bml_threshold.h, [71](#)
/bml/src/C-interface/bml_trace.h, [72](#)
/bml/src/C-interface/bml_transpose.h, [73](#)
/bml/src/C-interface/bml_transpose_triangle.h, [74](#)
/bml/src/C-interface/bml_types.h, [75](#)
/bml/src/C-interface/bml_types_private.h, [77](#)
/bml/src/C-interface/bml_utilities.h, [77](#)

Add Functions (C interface), [31](#)
 bml_add, [31](#)
 bml_add_identity, [31](#)
 bml_add_norm, [32](#)
 bml_scale_add_identity, [32](#)

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

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, [25](#)
 bml_default_domain, [25](#)
 bml_free_memory, [26](#)
 bml_free_ptr, [26](#)
 bml_identity_matrix, [26](#)
 bml_noinit_allocate_memory, [27](#)
 bml_noinit_matrix, [27](#)
 bml_noinit_rectangular_matrix, [28](#)
 bml_random_matrix, [28](#)

 bml_reallocate_memory, [29](#)
 bml_update_domain_matrix, [29](#)
 bml_zero_matrix, [30](#)

Allocation and Deallocation Functions (Fortran interface), [35](#)

bml_add
 Add Functions (C interface), [31](#)

bml_add_identity
 Add Functions (C interface), [31](#)

bml_add_norm
 Add Functions (C interface), [32](#)

bml_adjacency
 bml_submatrix.h, [68](#)

bml_adjacency_group
 bml_submatrix.h, [69](#)

bml_adjungate_triangle
 bml_adjungate_triangle.h, [44](#)

bml_adjungate_triangle.h
 bml_adjungate_triangle, [44](#)

bml_allGatherVParallel
 bml_parallel.h, [65](#)

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, [45](#)

bml_convert.h
 bml_convert, [45](#)

bml_copy
 bml_copy.h, [46](#)

bml_copy.h
 bml_copy, [46](#)
 bml_copy_new, [46](#)
 bml_reorder, [47](#)
 bml_restore_domain, [47](#)
 bml_save_domain, [47](#)

bml_copy_new
 bml_copy.h, [46](#)

bml_deallocate

- Allocation and Deallocation Functions (C interface), 25
- bml_deallocate_domain
 - Allocation and Deallocation Functions (C interface), 25
- bml_default_domain
 - Allocation and Deallocation Functions (C interface), 25
- bml_dense_order_t
 - bml_types.h, 76
- bml_distribution_mode_t
 - bml_types.h, 76
- bml_domain_t, 39
 - globalRowExtent, 39
 - globalRowMax, 39
 - globalRowMin, 40
 - localDispl, 40
 - localElements, 40
 - localRowExtent, 40
 - localRowMax, 40
 - localRowMin, 40
 - maxLocalExtent, 40
 - minLocalExtent, 40
 - totalCols, 41
 - totalProcs, 41
 - totalRows, 41
- bml_element_multiply.h
 - bml_element_multiply_AB, 48
- bml_element_multiply_AB
 - bml_element_multiply.h, 48
- bml_export_to_dense
 - Converting between Matrix Formats (C interface), 33
- bml_fnorm
 - bml_norm.h, 61
- bml_fnorm2
 - bml_norm.h, 61
- bml_free_memory
 - Allocation and Deallocation Functions (C interface), 26
- bml_free_ptr
 - Allocation and Deallocation Functions (C interface), 26
- bml_gershgorin
 - bml_normalize.h, 63
- bml_gershgorin_partial
 - bml_normalize.h, 64
- bml_get_bandwidth
 - bml_introspection.h, 51
- bml_get_deep_type
 - bml_introspection.h, 52
- bml_get_diagonal
 - bml_getters.h, 49
- bml_get_distribution_mode
 - bml_introspection.h, 52
- bml_get_element
 - bml_getters.h, 49
- bml_get_M
 - bml_introspection.h, 52
- bml_get_N
 - bml_introspection.h, 54
- bml_get_precision
 - bml_introspection.h, 54
- bml_get_row
 - bml_getters.h, 49
- bml_get_row_bandwidth
 - bml_introspection.h, 54
- bml_get_sparsity
 - bml_introspection.h, 55
- bml_get_type
 - bml_introspection.h, 55
- bml_getMyRank
 - bml_parallel.h, 65
- bml_getNRanks
 - bml_parallel.h, 65
- bml_getters.h
 - bml_get_diagonal, 49
 - bml_get_element, 49
 - bml_get_row, 49
- bml_group_matrix
 - bml_submatrix.h, 69
- bml_identity_matrix
 - Allocation and Deallocation Functions (C interface), 26
- bml_import_from_dense
 - Converting between Matrix Formats (C interface), 33
- bml_init
 - bml_init.h, 50
- bml_init.h
 - bml_init, 50
 - bml_initF, 51
- bml_initF
 - bml_init.h, 51
- bml_introspection.h
 - bml_get_bandwidth, 51
 - bml_get_deep_type, 52
 - bml_get_distribution_mode, 52
 - bml_get_M, 52
 - bml_get_N, 54
 - bml_get_precision, 54
 - bml_get_row_bandwidth, 54
 - bml_get_sparsity, 55
 - bml_get_type, 55
- bml_log
 - bml_logger.h, 57
- BML_LOG_DEBUG
 - bml_logger.h, 57
- BML_LOG_ERROR
 - bml_logger.h, 57
- BML_LOG_INFO
 - bml_logger.h, 57
- bml_log_level_t
 - bml_logger.h, 57
- bml_log_location
 - bml_logger.h, 58

- BML_LOG_WARNING
 - bml_logger.h, 57
- bml_logger.h
 - bml_log, 57
 - BML_LOG_DEBUG, 57
 - BML_LOG_ERROR, 57
 - BML_LOG_INFO, 57
 - bml_log_level_t, 57
 - bml_log_location, 58
 - BML_LOG_WARNING, 57
 - LOG_DEBUG, 56
 - LOG_ERROR, 56
 - LOG_INFO, 56
 - LOG_WARN, 57
- bml_matrix2submatrix
 - bml_submatrix.h, 70
- bml_matrix2submatrix_index
 - bml_submatrix.h, 70
- bml_matrix2submatrix_index_graph
 - bml_submatrix.h, 70
- bml_matrix_dimension_t, 41
 - bsizes, 42
 - N_cols, 42
 - N_nz_max, 42
 - N_rows, 42
 - NB, 42
- bml_matrix_precision_t
 - bml_types.h, 76
- bml_matrix_t
 - bml_types.h, 75
- bml_matrix_type_t
 - bml_types.h, 77
- bml_multiply
 - bml_multiply.h, 58
- bml_multiply.h
 - bml_multiply, 58
 - bml_multiply_AB, 59
 - bml_multiply_adjust_AB, 59
 - bml_multiply_x2, 60
- bml_multiply_AB
 - bml_multiply.h, 59
- bml_multiply_adjust_AB
 - bml_multiply.h, 59
- bml_multiply_x2
 - bml_multiply.h, 60
- bml_noinit_allocate_memory
 - Allocation and Deallocation Functions (C interface), 27
- bml_noinit_matrix
 - Allocation and Deallocation Functions (C interface), 27
- bml_noinit_rectangular_matrix
 - Allocation and Deallocation Functions (C interface), 28
- bml_norm.h
 - bml_fnorm, 61
 - bml_fnorm2, 61
 - bml_sum_AB, 61
 - bml_sum_squares, 62
 - bml_sum_squares2, 62
 - bml_sum_squares_submatrix, 63
- bml_normalize
 - bml_normalize.h, 64
- bml_normalize.h
 - bml_gershgorin, 63
 - bml_gershgorin_partial, 64
 - bml_normalize, 64
- bml_parallel.h
 - bml_allGatherVParallel, 65
 - bml_getMyRank, 65
 - bml_getNRanks, 65
- bml_print_bml_matrix
 - bml_utilities.h, 78
- bml_print_bml_vector
 - bml_utilities.h, 78
- bml_print_dense_matrix
 - bml_utilities.h, 79
- bml_print_dense_vector
 - bml_utilities.h, 79
- bml_random_matrix
 - Allocation and Deallocation Functions (C interface), 28
- bml_read_bml_matrix
 - bml_utilities.h, 79
- bml_reallocate_memory
 - Allocation and Deallocation Functions (C interface), 29
- bml_reorder
 - bml_copy.h, 47
- bml_restore_domain
 - bml_copy.h, 47
- bml_save_domain
 - bml_copy.h, 47
- bml_scale
 - bml_scale.h, 66
- bml_scale.h
 - bml_scale, 66
 - bml_scale_inplace, 66
 - bml_scale_new, 66
- bml_scale_add_identity
 - Add Functions (C interface), 32
- bml_scale_inplace
 - bml_scale.h, 66
- bml_scale_new
 - bml_scale.h, 66
- bml_shutdown
 - bml_shutdown.h, 67
- bml_shutdown.h
 - bml_shutdown, 67
 - bml_shutdownF, 68
- bml_shutdownF
 - bml_shutdown.h, 68
- bml_submatrix.h
 - bml_adjacency, 68
 - bml_adjacency_group, 69
 - bml_group_matrix, 69

- bml_matrix2submatrix, [70](#)
 - bml_matrix2submatrix_index, [70](#)
 - bml_matrix2submatrix_index_graph, [70](#)
 - bml_submatrix2matrix, [71](#)
- bml_submatrix2matrix
 - bml_submatrix.h, [71](#)
- bml_sum_AB
 - bml_norm.h, [61](#)
- bml_sum_squares
 - bml_norm.h, [62](#)
- bml_sum_squares2
 - bml_norm.h, [62](#)
- bml_sum_squares_submatrix
 - bml_norm.h, [63](#)
- bml_threshold
 - bml_threshold.h, [72](#)
- bml_threshold.h
 - bml_threshold, [72](#)
 - bml_threshold_new, [72](#)
- bml_threshold_new
 - bml_threshold.h, [72](#)
- bml_trace
 - bml_trace.h, [73](#)
- bml_trace.h
 - bml_trace, [73](#)
 - bml_trace_mult, [73](#)
- bml_trace_mult
 - bml_trace.h, [73](#)
- bml_transpose
 - bml_transpose.h, [74](#)
- bml_transpose.h
 - bml_transpose, [74](#)
 - bml_transpose_new, [74](#)
- bml_transpose_new
 - bml_transpose.h, [74](#)
- bml_transpose_triangle
 - bml_transpose_triangle.h, [75](#)
- bml_transpose_triangle.h
 - bml_transpose_triangle, [75](#)
- bml_types.h
 - bml_dense_order_t, [76](#)
 - bml_distribution_mode_t, [76](#)
 - bml_matrix_precision_t, [76](#)
 - bml_matrix_t, [75](#)
 - bml_matrix_type_t, [77](#)
 - bml_vector_t, [76](#)
 - csr, [77](#)
 - dense, [77](#)
 - dense_column_major, [76](#)
 - dense_row_major, [76](#)
 - distributed, [76](#)
 - distributed2d, [77](#)
 - double_complex, [77](#)
 - double_real, [77](#)
 - ellblock, [77](#)
 - ellpack, [77](#)
 - graph_distributed, [76](#)
 - precision_uninitialized, [77](#)
 - sequential, [76](#)
 - single_complex, [77](#)
 - single_real, [77](#)
 - type_uninitialized, [77](#)
- bml_update_domain_matrix
 - Allocation and Deallocation Functions (C interface), [29](#)
- bml_utilities.h
 - bml_print_bml_matrix, [78](#)
 - bml_print_bml_vector, [78](#)
 - bml_print_dense_matrix, [79](#)
 - bml_print_dense_vector, [79](#)
 - bml_read_bml_matrix, [79](#)
 - bml_write_bml_matrix, [80](#)
- bml_vector_t
 - bml_types.h, [76](#)
- bml_write_bml_matrix
 - bml_utilities.h, [80](#)
- bml_zero_matrix
 - Allocation and Deallocation Functions (C interface), [30](#)
- bsizes
 - bml_matrix_dimension_t, [42](#)
- Converting between Matrix Formats (C interface), [33](#)
 - bml_export_to_dense, [33](#)
 - bml_import_from_dense, [33](#)
- Converting between Matrix Formats (Fortran interface), [37](#)
- csr
 - bml_types.h, [77](#)
- dense
 - bml_types.h, [77](#)
- dense_column_major
 - bml_types.h, [76](#)
- dense_row_major
 - bml_types.h, [76](#)
- distributed
 - bml_types.h, [76](#)
- distributed2d
 - bml_types.h, [77](#)
- double_complex
 - bml_types.h, [77](#)
- double_real
 - bml_types.h, [77](#)
- ellblock
 - bml_types.h, [77](#)
- ellpack
 - bml_types.h, [77](#)
- globalRowExtent
 - bml_domain_t, [39](#)
- globalRowMax
 - bml_domain_t, [39](#)
- globalRowMin
 - bml_domain_t, [40](#)
- graph_distributed

- [bml_types.h](#), [76](#)
- [localDispl](#)
 - [bml_domain_t](#), [40](#)
- [localElements](#)
 - [bml_domain_t](#), [40](#)
- [localRowExtent](#)
 - [bml_domain_t](#), [40](#)
- [localRowMax](#)
 - [bml_domain_t](#), [40](#)
- [localRowMin](#)
 - [bml_domain_t](#), [40](#)
- [LOG_DEBUG](#)
 - [bml_logger.h](#), [56](#)
- [LOG_ERROR](#)
 - [bml_logger.h](#), [56](#)
- [LOG_INFO](#)
 - [bml_logger.h](#), [56](#)
- [LOG_WARN](#)
 - [bml_logger.h](#), [57](#)
- [maxLocalExtent](#)
 - [bml_domain_t](#), [40](#)
- [minLocalExtent](#)
 - [bml_domain_t](#), [40](#)
- [N_cols](#)
 - [bml_matrix_dimension_t](#), [42](#)
- [N_nz_max](#)
 - [bml_matrix_dimension_t](#), [42](#)
- [N_rows](#)
 - [bml_matrix_dimension_t](#), [42](#)
- [NB](#)
 - [bml_matrix_dimension_t](#), [42](#)
- [precision_uninitialized](#)
 - [bml_types.h](#), [77](#)
- [sequential](#)
 - [bml_types.h](#), [76](#)
- [single_complex](#)
 - [bml_types.h](#), [77](#)
- [single_real](#)
 - [bml_types.h](#), [77](#)
- [totalCols](#)
 - [bml_domain_t](#), [41](#)
- [totalProcs](#)
 - [bml_domain_t](#), [41](#)
- [totalRows](#)
 - [bml_domain_t](#), [41](#)
- [type_uninitialized](#)
 - [bml_types.h](#), [77](#)