bml

1.3.1

Generated by Doxygen 1.8.17

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.

Chapter 2

README

2.1 Introduction

This website is intended to provide some guidance on how to get and install the bml library. LA-UR number LA-UR-∗∗17-27373∗∗.

The basic matrix library (bml) is a collection of various matrix data formats (for dense and sparse) and their associated algorithms for basic matrix operations. Application programming interfaces (API) are available for both C and FORTRAN. The current status of this library allows us to use two different formats for representing matrix data. Currently these formats are: dense, ELLPACK-R, ELLBLOCK, ELLSORT, and CSR. For information on how to use the BML library can be find in `BML-API`.

2.2 Mailing List

We are running the following mailing list for discussions on usage and features of the bml library:

- `bml`
  - `Subscribe`
  - `Archives`

2.3 Supported Matrix Formats

The bml library supports the following matrix formats:

- dense
- ELLPACK-R
- ELLSORT
- ELLBLOCK
- CSR

2.4 Binary Packages

We offer binary packages of the bml library in `RPM format` thanks to SUSE's OpenBuild Service and for Ubuntu in `DEB format`.

2.5 Build Instructions

The bml library is built with CMake. For convenience, we provide a shell script which goes through the necessary motions and builds the library, runs the tests, and installs it (in the `install` directory).

2.5.1 For a quick installation

We suggest to take a look at the `example_build.sh` script that sets the most important environmental variables needed by `build.sh` script. Change the Variables according to the compilers and architecture. The script can be run just by doing:

```
$ ./example_build.sh
```

2.5.2 For a more involved installation

By running:

```
$ ./build.sh install
```

the library will be built in the `build` directory and installed in the `install` directory. In case you change any sources and simply want to rebuild the library, you don't have to run `build.sh` again, but rather

```
$ make -C build
```

The compiled library can be installed by running

```
$ make -C build install
```

The install directory can be modified by running

```
$ CMAKE_INSTALL_PREFIX=/some/path ./build.sh install
```

(which assumes that you are using the bash shell).

To build with GNU compilers, OpenMP, and Intel MKL do the following.

```
$ CC=gcc FC=gfortran \
    BLAS_VENDOR=Intel CMAKE_BUILD_TYPE=Release \
    BML_OPENMP=yes CMAKE_INSTALL_PREFIX=/some/path \
    ./build.sh install
```

To build with MPI, OpenMP, and use Intel MKL do the following.

```
$ CC=mpicc FC=mpif90 \
    BLAS_VENDOR=Intel CMAKE_BUILD_TYPE=Release \
    BML_OPENMP=yes BML_MPI=yes CMAKE_INSTALL_PREFIX=/some/path \
    ./build.sh install
```

2.5.3 Prerequisites

In order to build the library, the following tools need to be installed:

- `gcc` with Fortran support
- `>=cmake-2.8.8`
- `>=python-2.7`
- `>=OpenMP-3.1` (i.e. `>=gcc-4.7`)

2.5.4 If the build fails

In case the build fails for some reason, please contact the developers by opening an issue on GitHub (`https://github.com/lanl/bml/issues`) and attach the files

```
build/CMakeFiles/CMakeOutput.log
build/CMakeFiles/CMakeError.log
```

2.6 Developer Suggested Workflow

Our main development happens on the `master` branch and is continuously verified for correctness. If you would like to contribute with your work to the bml project, please follow the instructions at the GitHub help page `"About pull requests"`. To summarize:

- Fork the project on github
- Clone that forked repository
- Create a branch in it
- Commit any changes to the branch
- Push the branch to your forked repository
- Go to `https://github.com/lanl/bml` and click on 'Create Pull Request'

During the review process you might want to update your pull request. Please add commits or `amend` your existing commits as necessary. If you amend any commits you need to add the `--force-with-lease` option to the `git push` command. Please make sure that your pull request contains only one logical change (see `"Structural split of change"` for further details.

2.6.1 Coding Style

Please indent your C code using

```
$ indent -gnu -nut -i4 -bli0 -cli4 -ppi0 -cbi0 -npcs -bfda
```

You can use the script `indent.sh` to indent all C code.

2.7 Citing

If you find this library useful, we encourage you to cite us. Our project has a citable DOI:

with the following `bibtex` snipped:

```
@misc{bml,
  author        = {Nicolas Bock and
                   Susan Mniszewski and
                   Bálint Aradi and
                   Michael Wall and
                   Christian F. A. Negre
                   Jamal Mohd-Yusof and
                   Anders N. M. Niklasson},
  title         = {qmmd/bml v1.2.3},
  month         = feb,
  year          = 2018,
  doi           = {10.5281/zenodo.841949},
  url           = {https://doi.org/10.5281/zenodo.841949}
}
```

Another citation source is the following journal article: `BMLPaper`

2.8 Authors

The core developers of the bml in alphabetical order:

- Christian Negre `cnegre@lanl.gov`
- Nicolas Bock `nicolasbock@gmail.com`
- Susan M. Mniszewski `smm@lanl.gov`

2.9 Contributors

- Adedoyin Adetokunbo `aadedoyin@lanl.gov`
- Bálint Aradi `aradi@uni-bremen.de`
- Daniel Osei-Kuffuor `oseikuffuor1@llnl.gov`
- Jamaludin Mohd-Yusof `jamal@lanl.gov`
- Jean-Luc Fattebert `fattebertj@ornl.gov`
- Mike Wall `mewall@lanl.gov`

2.10 License

The bml library is licensed under the BSD 3-clause license.

2.11 LA-CC

NOTICE OF OSS COPYRIGHT ASSERTION:

LANS has asserted copyright on the software package entitled *Basic Matrix Library (bml), Version 0.x (C16006).*

2.11.1 ABSTRACT

The basic matrix library (bml) is a collection of various matrix data formats (for dense and sparse) and their associated algorithms for basic matrix operations.

This code is unclassified and has been assigned LA-CC-∗∗15-093∗∗. Los Alamos National Laboratory's Export Control Team made an in-house determination that this software is controlled under Department of Commerce regulations and the Export Control Classification Number (ECCN) **EAR99**. The export control review is attached.

The developers intend to distribute this software package under the OSI Certified **BSD 3-Clause License** (`http↩ ://www.opensource.org/licenses/BSD-3-Clause`)

This code was developed using funding from:

- Basic Energy Sciences (LANL2014E8AN) and the Laboratory Directed Research and Development Program of Los Alamos National Laboratory. To tests these developments we used resources provided by the Los Alamos National Laboratory Institutional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Administration
- Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

Larry Kwei, LAFO Program Manager, has granted his concurrence to asserting copyright and then distributing the **Basic Matrix Library (bml), Version 0.x** code using an open source software license. See attached memo.

LANS acknowledges that it will comply with the DOE OSS policy as follows:

1. submit form DOE F 241.4 to the Energy Science and Technology Software Center (ESTSC),
2. provide the unique URL on the form for ESTSC to distribute, and
3. maintain an OSS Record available for inspection by DOE.

Following is a table briefly summarizes information for this software package:

CODE NAME	Basic Matrix Library (bml), Version 0.x (C16006)
Classification Review Number	**LA-CC-15-093**
Export Control Classification Number (ECCN)	**EAR99**
B&R Code	**YN0100000**

Chapter 3

Future Plans

3.1 Matrix Types

Support types:

- bml_matrix_t
- Colinear
- Noncolinear

3.2 Precisions

The bml supports the following precisions:

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

3.3 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
  - $\mathrm{Tr}[A]$: bml_trace::bml_trace
  - $\mathrm{Tr}[AB]$: 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 4

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 5

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 6

Developer Documentation

6.1 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.

6.2 Coding Style

Please indent your C code using

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

Back to the main page.

Chapter 7

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 varibles 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.

7.1 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 8

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 replace 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_↩ TEST. 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 fuctionalities 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.

8.1 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
```

8.2 ADDING A FORTRAN TEST

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

Chapter 10

Class Index

10.1 Class List

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

Chapter 11

File Index

11.1 File List

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

Chapter 12

Module Documentation

12.1 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)

12.1.1 Detailed Description

12.1.2 Function Documentation

12.1.2.1 bml_allocate_memory()

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

Allocate and zero a chunk of memory.

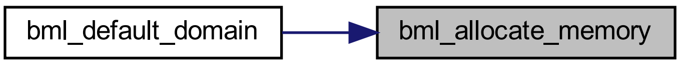
Parameters

size	The size of the memory.

Returns

A pointer to the allocated chunk.

Here is the caller graph for this function:



12.1.2.2 bml_allocated()

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

Check if matrix is allocated.

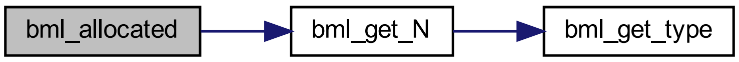
Parameters

A[in,out]	Matrix

Returns

> 0 if allocated, else -1

Here is the call graph for this function:



12.1.2.3 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

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

Returns

The matrix.

12.1.2.4 bml_clear()

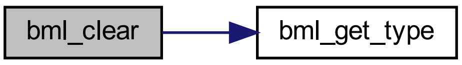
```
void bml_clear (
             bml_matrix_t * A )
```

Clear a matrix.

Parameters

A[in,out]	The matrix.

Here is the call graph for this function:



12.1.2.5 bml_deallocate()

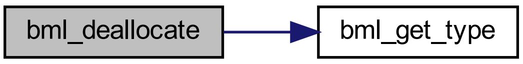
```
void bml_deallocate (
            bml_matrix_t ** A )
```

Deallocate a matrix.

Parameters

A[in,out]	The matrix.

Here is the call graph for this function:



12.1.2.6 bml_deallocate_domain()

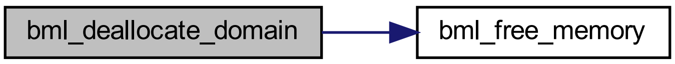
```
void bml_deallocate_domain (
            bml_domain_t * D )
```

Deallocate a domain.

Parameters

D[in,out]	The domain.

Here is the call graph for this function:



12.1.2.7 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

N	The number of rows
M	The number of columns
distrib_mode	The distribution mode

Returns

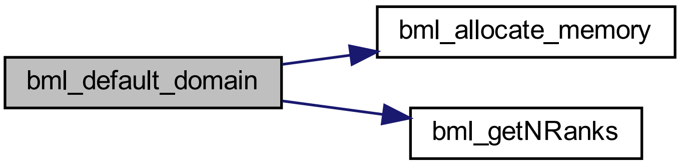
The domain

For first rank

For middle ranks

For last rank

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



12.1.2.8 bml_free_memory()

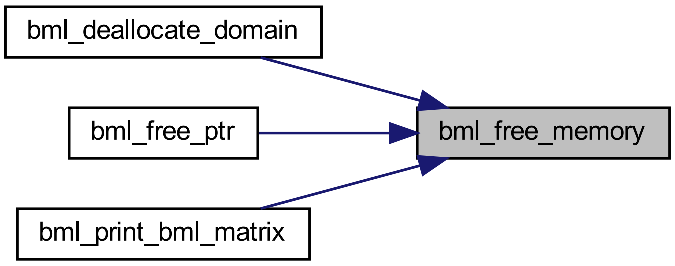
```
void bml_free_memory (
             void * ptr )
```

Deallocate a chunk of memory.

Parameters

ptr	A pointer to the previously allocated chunk.

Here is the caller graph for this function:



12.1.2.9 bml_free_ptr()

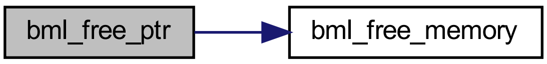
```
void bml_free_ptr (
            void ** ptr )
```

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

ptr	A pointer to the previously allocated chunk.

Here is the call graph for this function:



12.1.2.10 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

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

Returns

The matrix.

12.1.2.11 bml_noinit_allocate_memory()

```
void* bml_noinit_allocate_memory (
            size_t size )
```

Allocate a chunk of memory without initialization.

Parameters

size	The size of the memory.

Returns

A pointer to the allocated chunk.

12.1.2.12 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.

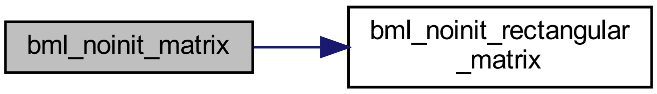
Parameters

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

Returns

The matrix.

Here is the call graph for this function:



12.1.2.13 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.

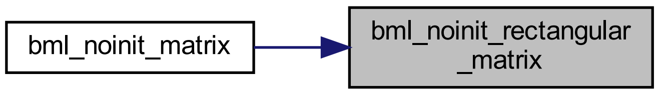
Parameters

matrix_type	The matrix type.
matrix_precision	The precision of the matrix.
matrix_dimension	The matrix size.
distrib_mode	The distribution mode.

Returns

The matrix.

Here is the caller graph for this function:



12.1.2.14 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

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

Returns

The matrix.

12.1.2.15 bml_reallocate_memory()

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

Reallocate a chunk of memory.

Parameters

size	The size of the memory.

Returns

A pointer to the reallocated chunk.

12.1.2.16 bml_update_domain()

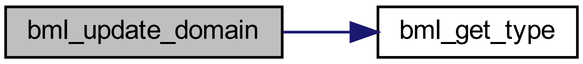
```
void bml_update_domain (
            bml_matrix_t * A,
            int * localPartMin,
            int * localPartMax,
            int * nnodesInPart )
```

Update a domain for a bml matrix.

Parameters

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

Here is the call graph for this function:



12.1.2.17 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

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

Returns

The matrix.

12.2 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)

12.2.1 Detailed Description

12.2.2 Function Documentation

12.2.2.1 bml_add()

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

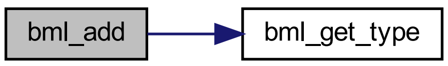
Matrix addition.

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

Parameters

A	Matrix A
B	Matrix B
alpha	Scalar factor multiplied by A
beta	Scalar factor multiplied by B
threshold	Threshold for matrix addition

Here is the call graph for this function:



12.2.2.2 bml_add_identity()

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

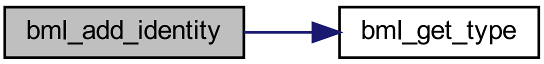
Matrix addition.

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

Parameters

A	Matrix A
beta	Scalar factor multiplied by I
threshold	Threshold for matrix addition

Here is the call graph for this function:



12.2.2.3 bml_add_norm()

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

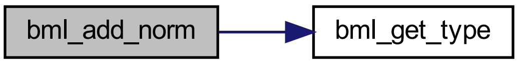
Matrix addition with calculation of TrNorm.

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

Parameters

A	Matrix A
B	Matrix B
alpha	Scalar factor multiplied by A
beta	Scalar factor multiplied by B
threshold	Threshold for matrix addition

Here is the call graph for this function:



12.2.2.4 bml_scale_add_identity()

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

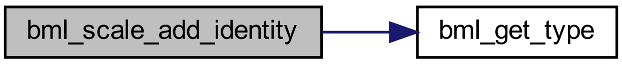
Matrix addition.

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

Parameters

A	Matrix A
alpha	Scalar factor multiplied by A
beta	Scalar factor multiplied by I
threshold	Threshold for matrix addition

Here is the call graph for this function:



12.3 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 distrib↩ _mode)

12.3.1 Detailed Description

12.3.2 Function Documentation

12.3.2.1 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);
```

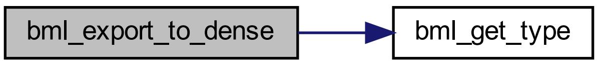
Parameters

A	The bml matrix
order	The matrix element order

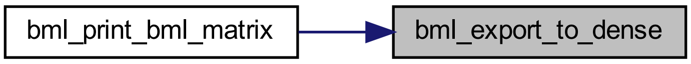
Returns

The dense matrix

Here is the call graph for this function:



Here is the caller graph for this function:



12.3.2.2 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

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

Returns

The bml matrix

12.4 Allocation and Deallocation Functions (Fortran interface)

12.5 Add Functions (Fortran interface)

12.6 Converting between Matrix Formats (Fortran interface)

Chapter 13

Class Documentation

13.1 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

13.1.1 Detailed Description

Decomposition for working in parallel.

13.1.2 Member Data Documentation

13.1.2.1 globalRowExtent

```
int bml_domain_t::globalRowExtent
```

global total rows

13.1.2.2 globalRowMax

```
int bml_domain_t::globalRowMax
```

global maximum row number

13.1.2.3 globalRowMin

```
int bml_domain_t::globalRowMin
```

global minimum row number

13.1.2.4 localDispl

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

local displacements per rank for 2D

13.1.2.5 localElements

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

local number of elements per rank

13.1.2.6 localRowExtent

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

extent of rows per rank, localRowMax - localRowMin

13.1.2.7 localRowMax

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

maximum row per rank

13.1.2.8 localRowMin

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

minimum row per rank

13.1.2.9 maxLocalExtent

```
int bml_domain_t::maxLocalExtent
```

maximum extent for most processors

13.1.2.10 minLocalExtent

```
int bml_domain_t::minLocalExtent
```

minimum extent for last processors

13.1.2.11 totalCols

```
int bml_domain_t::totalCols
```

total number of columns

13.1.2.12 totalProcs

```
int bml_domain_t::totalProcs
```

number of processors

13.1.2.13 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

13.2 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

13.2.1 Detailed Description

The matrix dimensions.

13.2.2 Member Data Documentation

13.2.2.1 bsizes

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

The block sizes (for block_ellpack).

13.2.2.2 N_cols

```
int bml_matrix_dimension_t::N_cols
```

The number of columns.

13.2.2.3 N_nz_max

```
int bml_matrix_dimension_t::N_nz_max
```

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

13.2.2.4 N_rows

```
int bml_matrix_dimension_t::N_rows
```

The number of rows.

13.2.2.5 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

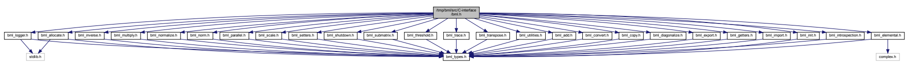
Chapter 14

File Documentation

14.1 /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_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:



14.1.1 Detailed Description

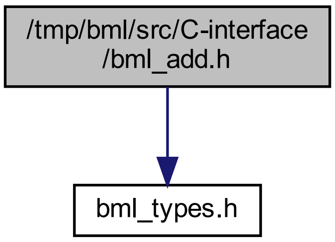
Copyright

Los Alamos National Laboratory 2015

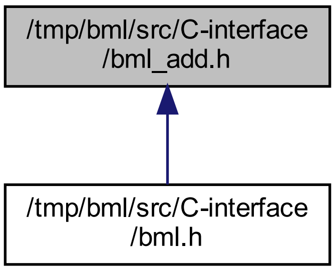
14.2 /tmp/bml/src/C-interface/bml_add.h File Reference

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

Include dependency graph for bml_add.h:



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



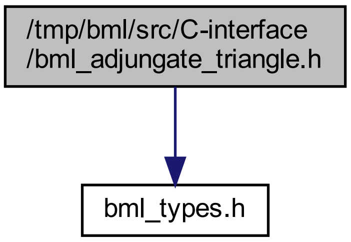
Functions

- 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)

14.3 /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)

14.3.1 Function Documentation

14.3.1.1 bml_adjungate_triangle()

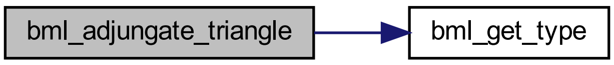
```
void bml_adjungate_triangle (
            bml_matrix_t * A,
            char * triangle )
```

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

Parameters

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

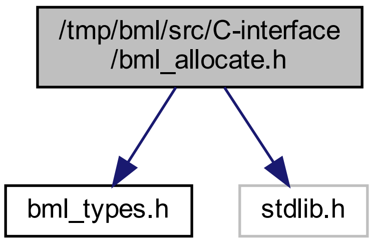
Here is the call graph for this function:



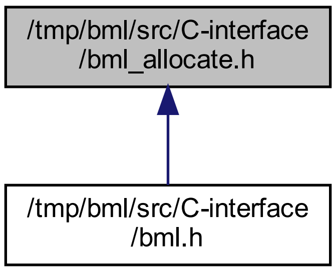
14.4 /tmp/bml/src/C-interface/bml_allocate.h File Reference

```
#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

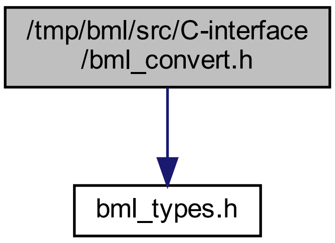
- int bml_allocated (bml_matrix_t ∗A)
- void ∗ bml_allocate_memory (size_t s)
- void ∗ bml_noinit_allocate_memory (size_t s)
- void ∗ bml_reallocate_memory (void ∗ptr, const size_t size)
- void bml_free_memory (void ∗ptr)
- void bml_free_ptr (void ∗∗ptr)
- void bml_deallocate (bml_matrix_t ∗∗A)
- void 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)

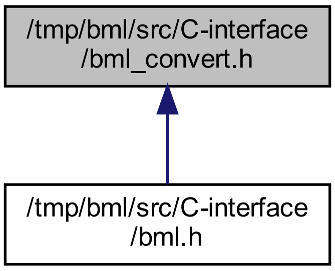
14.5 /tmp/bml/src/C-interface/bml_convert.h File Reference

```
#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)

14.5.1 Function Documentation

14.5.1.1 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.

$A \to B$

Parameters

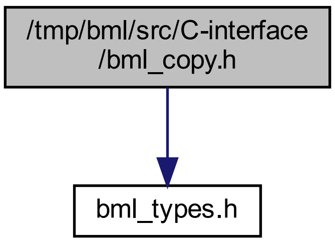
A	The input matrix.

Returns

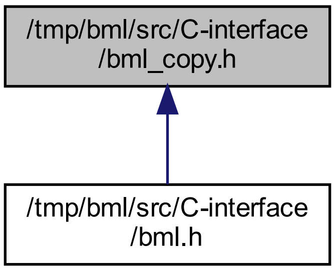
The converted matrix B.

14.6 /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)

14.6.1 Function Documentation

14.6.1.1 bml_copy()

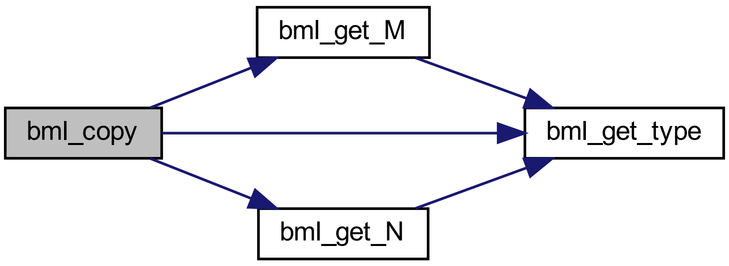
```
void bml_copy (
            bml_matrix_t ∗ A,
            bml_matrix_t ∗ B )
```

Copy a matrix.

Parameters

A	Matrix to copy
B	Copy of Matrix A

Here is the call graph for this function:



14.6.1.2 bml_copy_domain()

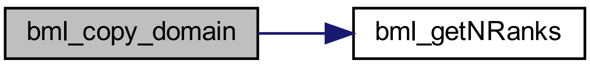
```
void bml_copy_domain (
            bml_domain_t * A,
            bml_domain_t * B )
```

Copy a domain.

Parameters

A	Domain to copy
B	Copy of Domain A

Here is the call graph for this function:



14.6.1.3 bml_copy_new()

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

Copy a matrix - result is a new matrix.

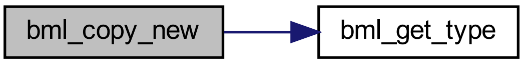
Parameters

A	Matrix to copy

Returns

A Copy of A

Here is the call graph for this function:



14.6.1.4 bml_reorder()

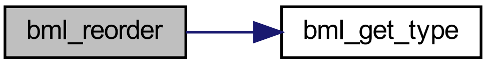
```
void bml_reorder (
            bml_matrix_t * A,
            int * perm )
```

Reorder a matrix in place.

Parameters

A	Matrix to reorder
perm	permutation vector for reordering

Here is the call graph for this function:



14.6.1.5 bml_restore_domain()

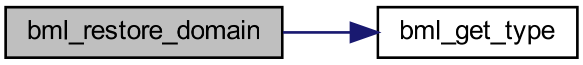
```
void bml_restore_domain (
             bml_matrix_t * A )
```

Restore to saved domain for bml matrix.

Parameters

A	Matrix with domain

Here is the call graph for this function:



14.6.1.6 bml_save_domain()

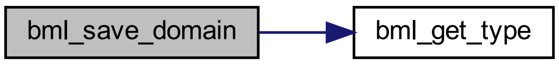
```
void bml_save_domain (
             bml_matrix_t * A )
```

Save current domain for bml matrix.

Parameters

A	Matrix with domain

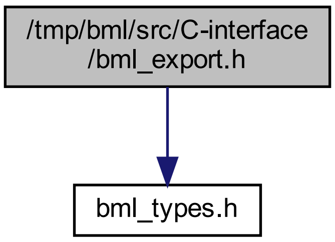
Here is the call graph for this function:



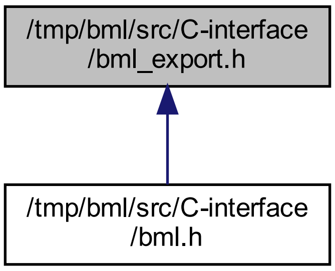
14.7 /tmp/bml/src/C-interface/bml_export.h File Reference

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

Include dependency graph for bml_export.h:



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



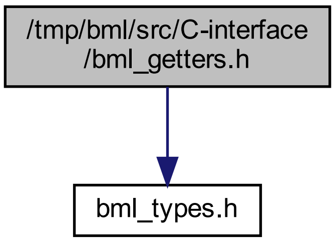
Functions

- void ∗ bml_export_to_dense (bml_matrix_t ∗A, bml_dense_order_t order)

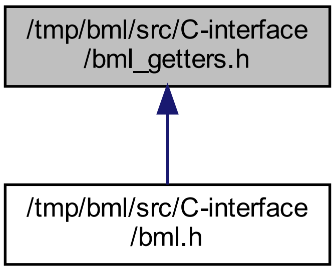
14.8 /tmp/bml/src/C-interface/bml_getters.h File Reference

```
#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)

14.8.1 Function Documentation

14.8.1.1 bml_get_diagonal()

```
void* bml_get_diagonal (
            bml_matrix_t * A )
```

Get the diagonal.

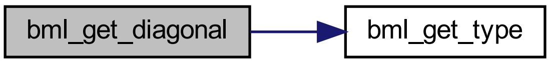
Parameters

A	The matrix.

Returns

The diagonal (an array)

Here is the call graph for this function:



14.8.1.2 bml_get_element()

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

Return a single matrix element.

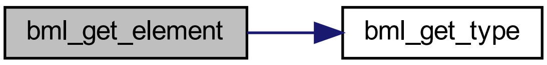
Parameters

i	The row index
j	The column index
A	The bml matrix

Returns

The matrix element

Here is the call graph for this function:



14.8.1.3 bml_get_row()

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

Get a whole row.

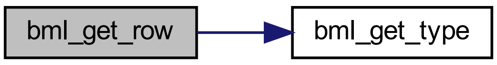
Parameters

A	The matrix.
i	The row index.

Returns

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

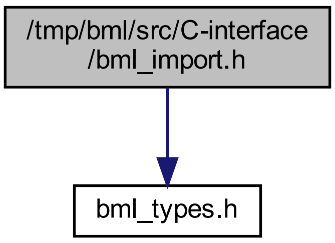
Here is the call graph for this function:



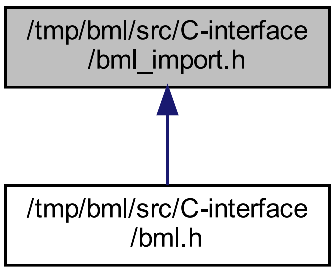
14.9 /tmp/bml/src/C-interface/bml_import.h File Reference

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

Include dependency graph for bml_import.h:



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



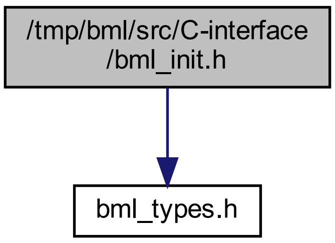
Functions

- 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 distrib↩
_mode)

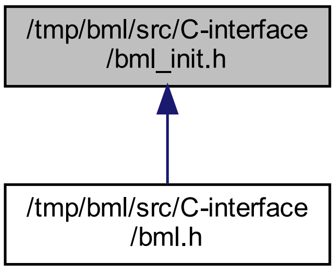
14.10 /tmp/bml/src/C-interface/bml_init.h File Reference

```
#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 (int ∗argc, char ∗∗∗argv)
- void bml_initF (int fcomm)

14.10.1 Function Documentation

14.10.1.1 bml_init()

```
void bml_init (
             int * argc,
             char *** argv )
```

Initialize.

Parameters

argc	Number of args
argv	Args

14.10.1.2 bml_initF()

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

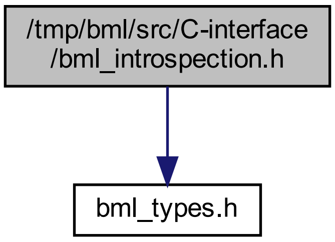
Initialize from Fortran.

Parameters

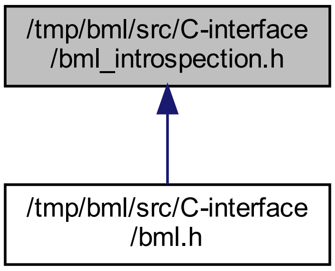
Comm	from Fortran

14.11 /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_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)

14.11.1 Function Documentation

14.11.1.1 bml_get_bandwidth()

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

Return the bandwidth of a matrix.

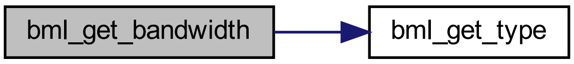
Parameters

A	The bml matrix.

Returns

The bandwidth of row i.

Here is the call graph for this function:



14.11.1.2 bml_get_distribution_mode()

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

Return the distribution mode of a matrix.

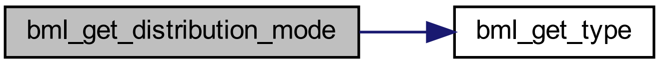
Parameters

A	The bml matrix.

Returns

The distibution mode of matrix A.

Here is the call graph for this function:



14.11.1.3 bml_get_M()

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

Return the matrix parameter M.

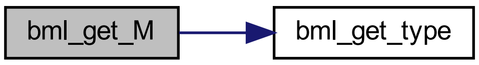
Parameters

A	The matrix.

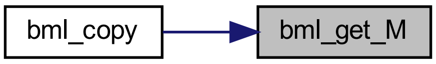
Returns

The matrix parameter M.

Here is the call graph for this function:



Here is the caller graph for this function:



14.11.1.4 bml_get_N()

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

Return the matrix size.

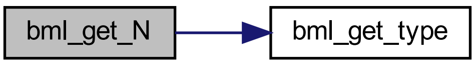
Parameters

A	The matrix.

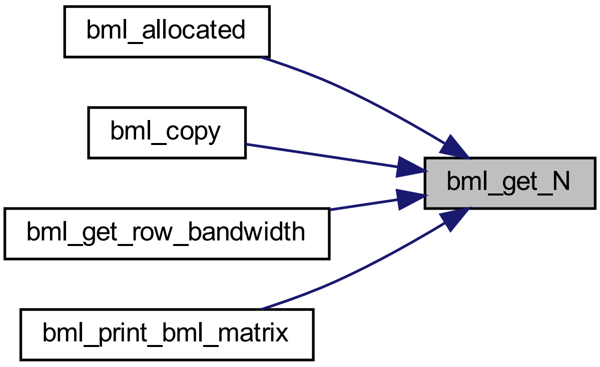
Returns

The matrix size.

Here is the call graph for this function:



Here is the caller graph for this function:



14.11.1.5 bml_get_precision()

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

Return the matrix precision.

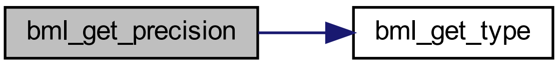
Parameters

A	The matrix.

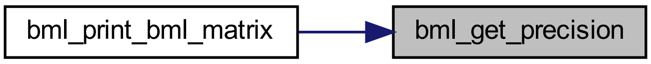
Returns

The matrix precision.

Here is the call graph for this function:



Here is the caller graph for this function:



14.11.1.6 bml_get_row_bandwidth()

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

Return the bandwidth of a row in the matrix.

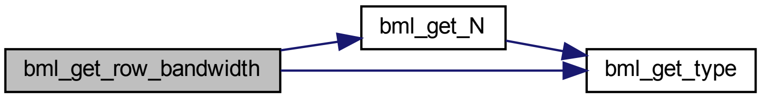
Parameters

A	The bml matrix.
i	The row index.

Returns

The bandwidth of row i.

Here is the call graph for this function:



14.11.1.7 bml_get_sparsity()

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

Return the sparsity of a matrix.

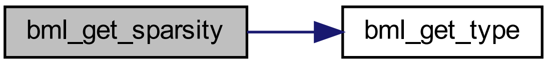
Parameters

A	The bml matrix.
threshold	The threshold used to compute the sparsity.

Returns

The sparsity of matrix A.

Here is the call graph for this function:



14.11.1.8 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 "unitialized" is returned.

Parameters

A	The matrix.

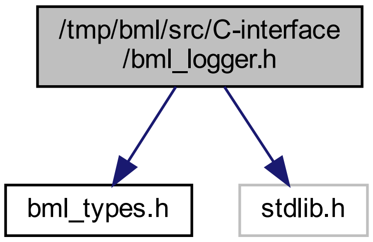
Returns

The matrix type.

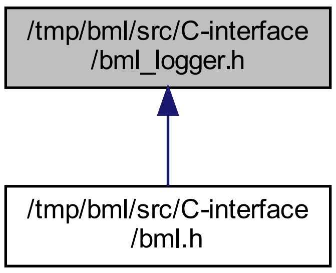
14.12 /tmp/bml/src/C-interface/bml_logger.h File Reference

```
#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 linenumber, char ∗format,...)

14.12.1 Macro Definition Documentation

14.12.1.1 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.

14.12.1.2 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.

14.12.1.3 LOG_INFO

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

Convenience macro to write a BML_LOG_INFO level message.

14.12.1.4 LOG_WARN

```
#define LOG_WARN(
              format,
              ...  ) bml_log_location(BML_LOG_WARNING, __FILE__, __LINE__, format, ##__VA_ARG↩
S__)
```

Convenience macro to write a BML_LOG_WARNING level message.

14.12.2 Enumeration Type Documentation

14.12.2.1 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.

14.12.3 Function Documentation

14.12.3.1 bml_log()

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

Log a message.

Parameters

log_level	The log level.
format	The format (as in printf()).

14.12.3.2 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..

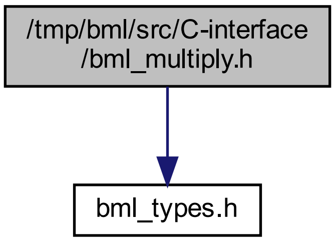
Parameters

log_level	The log level.
filename	The filename to log.
linenumber	The linenumber.
format	The format (as in printf()).

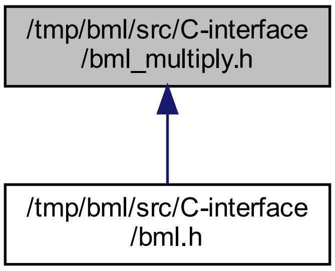
14.13 /tmp/bml/src/C-interface/bml_multiply.h File Reference

`#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)

14.13.1 Function Documentation

14.13.1.1 bml_multiply()

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

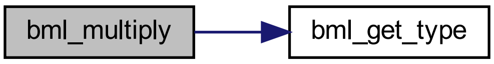
Matrix multiply.

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

Parameters

A	Matrix A
B	Matrix B
C	Matrix C
alpha	Scalar factor that multiplies A $*$ B
beta	Scalar factor that multiplies C
threshold	Threshold for multiplication

Here is the call graph for this function:



14.13.1.2 bml_multiply_AB()

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

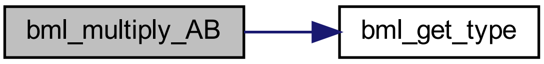
Matrix multiply.

C = A $*$ B

Parameters

A	Matrix A
B	Matrix B
C	Matrix C
threshold	Threshold for multiplication

Here is the call graph for this function:



14.13.1.3 bml_multiply_adjust_AB()

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

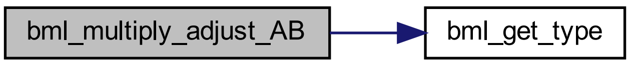
Matrix multiply with threshold adjustment.

C = A ∗ B

Parameters

A	Matrix A
B	Matrix B
C	Matrix C
threshold	Threshold for multiplication

Here is the call graph for this function:



14.13.1.4 bml_multiply_x2()

```
void* bml_multiply_x2 (
             bml_matrix_t * X,
             bml_matrix_t * X2,
             double threshold )
```

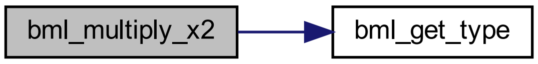
Matrix multiply.

$X^2 \leftarrow X\, X$

Parameters

X	Matrix X
X2	MatrixX2
threshold	Threshold for multiplication

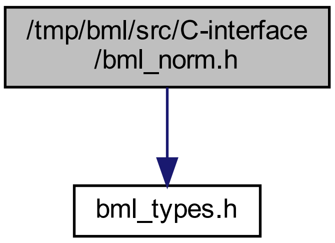
Here is the call graph for this function:



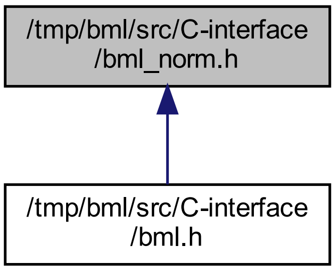
14.14 /tmp/bml/src/C-interface/bml_norm.h File Reference

```
#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_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)

14.14.1 Function Documentation

14.14.1.1 bml_fnorm()

```
double bml_fnorm (
            bml_matrix_t * A )
```

Calculate the Frobenius norm of a matrix.

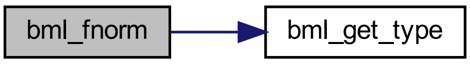
Parameters

A	Matrix A

Returns

Frobenius norm of Matrix A

Here is the call graph for this function:



14.14.1.2 bml_fnorm2()

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

Calculate the Frobenius norm of 2 matrices.

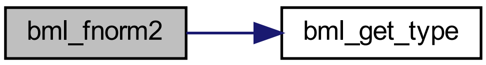
Parameters

A	Matrix A
B	Matrix B

Returns

Frobenius norm of Matrix A

Here is the call graph for this function:



14.14.1.3 bml_sum_squares()

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

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

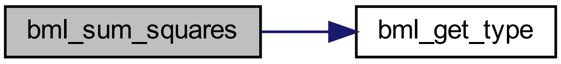
Parameters

A	Matrix A

Returns

sum of squares of all elements in A

Here is the call graph for this function:



14.14.1.4 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

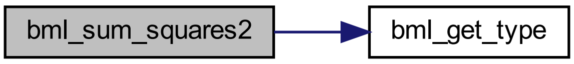
Parameters

A	Matrix
B	Matrix
alpha	Multiplier for matrix A
beta	Multiplier for matrix B
threshold	Threshold

Returns

sum of squares of alpha ∗ A + beta ∗ B

Here is the call graph for this function:



14.14.1.5 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.

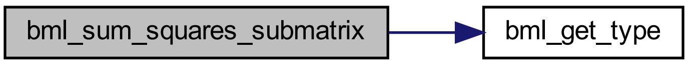
Parameters

A	Matrix A
core_pos	Core rows in A
core_size	Number of core rows

Returns

sum of squares of all elements in A

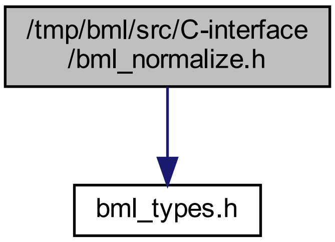
Here is the call graph for this function:



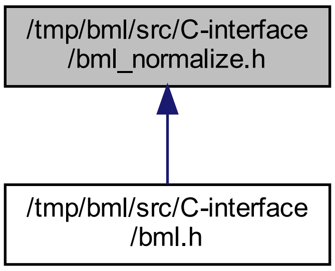
14.15 /tmp/bml/src/C-interface/bml_normalize.h File Reference

```
#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)

14.15.1 Function Documentation

14.15.1.1 bml_gershgorin()

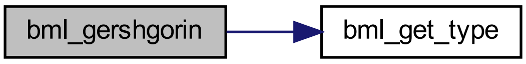
```
void* bml_gershgorin (
            bml_matrix_t * A )
```

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:



14.15.1.2 bml_gershgorin_partial()

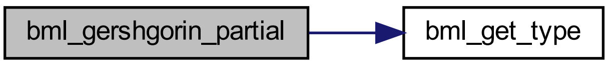
```
void* bml_gershgorin_partial (
            bml_matrix_t * A,
            int nrows )
```

Calculate Gershgorin bounds for partial matrix.

Parameters

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

Here is the call graph for this function:



14.15.1.3 bml_normalize()

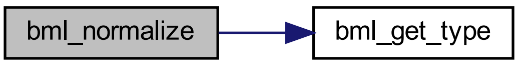
```
void bml_normalize (
            bml_matrix_t * A,
            double mineval,
            double maxeval )
```

Normalize matrix given Gershgorin bounds.

Parameters

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

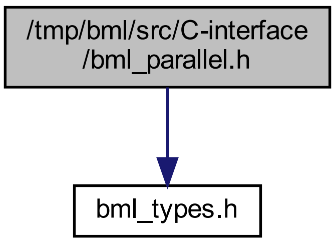
Here is the call graph for this function:



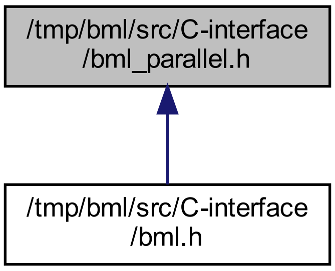
14.16 /tmp/bml/src/C-interface/bml_parallel.h File Reference

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

Include dependency graph for bml_parallel.h:



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


/tmp/bml/src/C-interface /bml_parallel.h
/tmp/bml/src/C-interface /bml.h

Functions

- int bml_getNRanks (void)
- int bml_getMyRank (void)
- void **bml_initParallelF** (int fcomm)
- void **bml_shutdownParallelF** ()
- int **bml_printRank** (void)
- void **bml_initParallel** (int ∗argc, char ∗∗∗argv)
- 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)

14.16.1 Function Documentation

14.16.1.1 bml_allGatherVParallel()

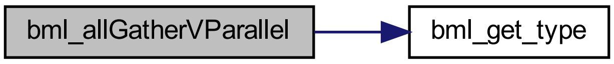
```
void bml_allGatherVParallel (
            bml_matrix_t * A )
```

Exchange pieces of matrix across MPI ranks.

Parameters

A	Matrix A

Here is the call graph for this function:



14.16.1.2 bml_getMyRank()

```
int bml_getMyRank (
            void  )
```

Get local MPI rank.

14.16.1.3 bml_getNRanks()

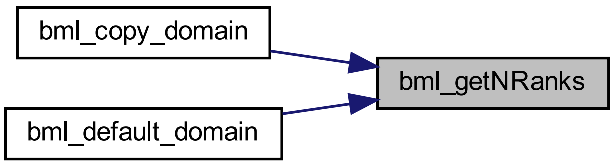
```
int bml_getNRanks (
            void  )
```

Initialize.

Parameters

argc	Number of args
argv	Args Get number of MPI ranks.

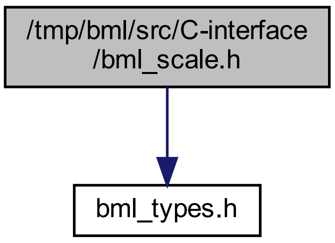
Here is the caller graph for this function:



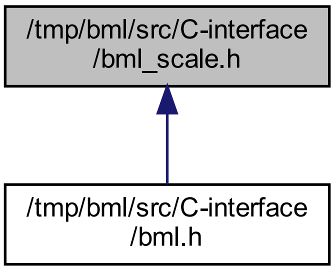
14.17 /tmp/bml/src/C-interface/bml_scale.h File Reference

```
#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)

14.17.1 Function Documentation

14.17.1.1 bml_scale()

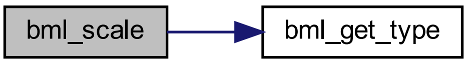
```
void bml_scale (
            void * scale_factor,
            bml_matrix_t * A,
            bml_matrix_t * B )
```

Scale a matrix - resulting matrix exists.

Parameters

scale_factor	Scale factor for A
A	Matrix to scale
B	Scaled Matrix

Here is the call graph for this function:



14.17.1.2 bml_scale_inplace()

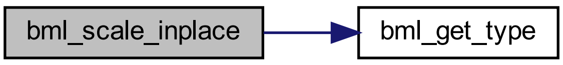
```
void bml_scale_inplace (
            void * scale_factor,
            bml_matrix_t * A )
```

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

Parameters

scale_factor	Scale factor for A
A	[inout] Matrix to scale

Here is the call graph for this function:



14.17.1.3 bml_scale_new()

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

Scale a matrix - resulting matrix is new.

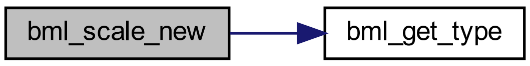
Parameters

scale_factor	Scale factor for A
A	Matrix to scale

Returns

A Scaled Copy of A

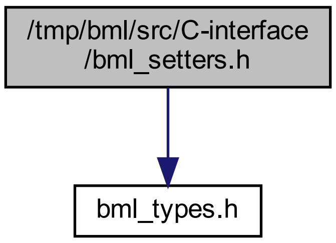
Here is the call graph for this function:



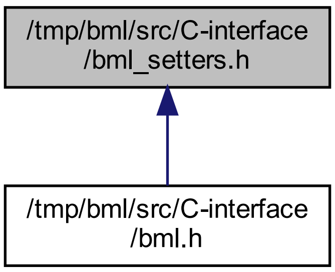
14.18 /tmp/bml/src/C-interface/bml_setters.h File Reference

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

Include dependency graph for bml_setters.h:



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



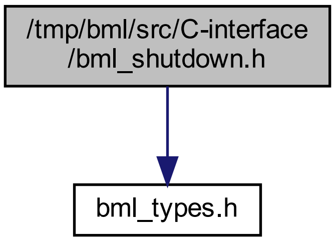
Functions

- 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)

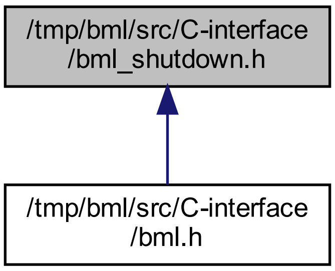
14.19 /tmp/bml/src/C-interface/bml_shutdown.h File Reference

```
#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 ()

14.19.1 Function Documentation

14.19.1.1 bml_shutdown()

```
void bml_shutdown ( )
```

Shutdown.

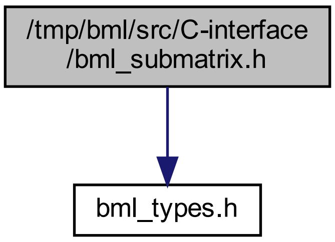
14.19.1.2 bml_shutdownF()

```
void bml_shutdownF ( )
```

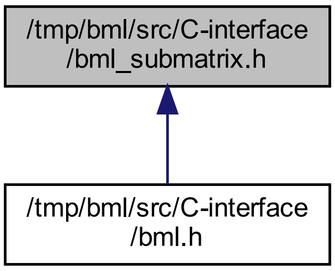
Shutdown from Fortran.

14.20 /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 llsize, 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)

14.20.1 Function Documentation

14.20.1.1 bml_adjacency()

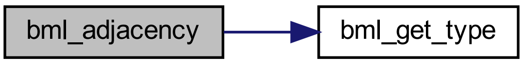
```
void bml_adjacency (
            bml_matrix_t * A,
            int * xadj,
            int * adjncy,
            int base_flag )
```

Assemble adjacency structures from matrix based on rows.

Parameters

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

Here is the call graph for this function:



14.20.1.2 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.

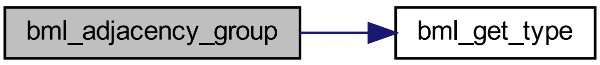
Parameters

A	Submatrix A
hindex	Index for each node element

Parameters

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

Here is the call graph for this function:



14.20.1.3 bml_group_matrix()

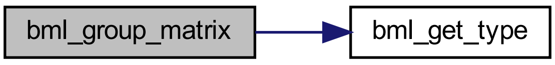
```
bml_matrix_t* bml_group_matrix (
              bml_matrix_t * A,
              int * hindex,
              int ngroups,
              double threshold )
```

Assemble matrix based on groups of rows from a matrix.

Parameters

A	Matrix A
hindex	Indeces of nodes
ngroups	Number of groups
threshold	Threshold for graph

Here is the call graph for this function:



14.20.1.4 bml_matrix2submatrix()

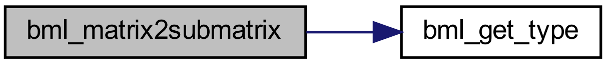
```
void bml_matrix2submatrix (
            bml_matrix_t * A,
            bml_matrix_t * B,
            int * core_halo_index,
            int lsize )
```

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

Parameters

A	Matrix A
B	Submatrix B
core_halo_index	Set of row indeces for submatrix
llsize	Number of indeces

Here is the call graph for this function:



14.20.1.5 bml_matrix2submatrix_index()

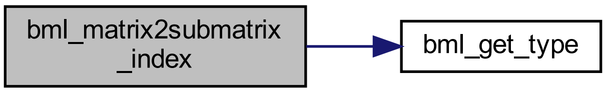
```
void bml_matrix2submatrix_index (
            bml_matrix_t * A,
            bml_matrix_t * B,
            int * nodelist,
            int nsize,
            int * core_halo_index,
            int * vsize,
            int double_jump_flag )
```

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

Parameters

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

Here is the call graph for this function:



14.20.1.6 bml_matrix2submatrix_index_graph()

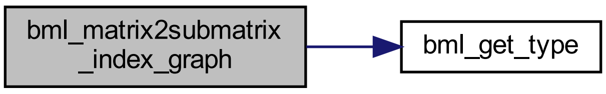
```
void bml_matrix2submatrix_index_graph (
            bml_matrix_t * B,
            int * nodelist,
            int nsize,
            int * core_halo_index,
            int * vsize,
            int double_jump_flag )
```

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

Parameters

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

Here is the call graph for this function:



14.20.1.7 bml_submatrix2matrix()

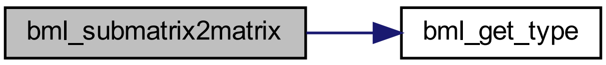
```
void bml_submatrix2matrix (
            bml_matrix_t * A,
            bml_matrix_t * B,
            int * core_halo_index,
            int lsize,
            int llsize,
            double threshold )
```

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

Parameters

A	Submatrix A
B	Matrix B
core_halo_index	Set of submatrix row indeces
lsize	Number of indeces
llsize	Number of core positions

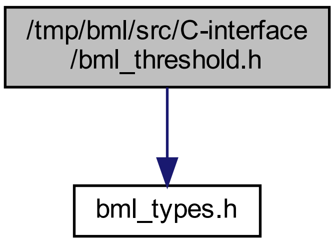
Here is the call graph for this function:



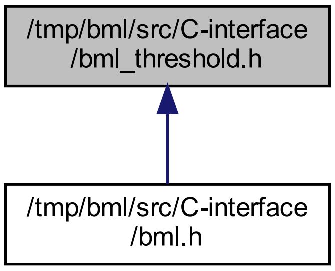
14.21 /tmp/bml/src/C-interface/bml_threshold.h File Reference

```
#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)

14.21.1 Function Documentation

14.21.1.1 bml_threshold()

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

Threshold matrix.

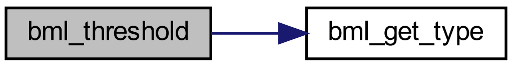
Parameters

A	Matrix to be thresholded
threshold	Threshold value

Returns

Thresholded A

Here is the call graph for this function:



14.21.1.2 bml_threshold_new()

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

Threshold matrix.

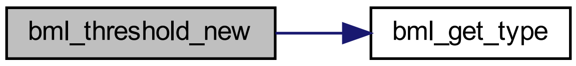
Parameters

A	Matrix to be thresholded
threshold	Threshold value

Returns

Thresholded A

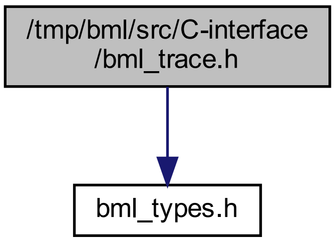
Here is the call graph for this function:



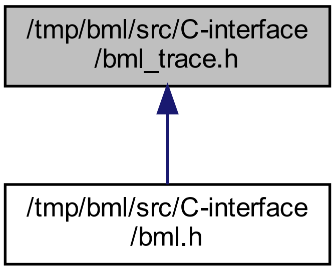
14.22 /tmp/bml/src/C-interface/bml_trace.h File Reference

```
#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)

14.22.1 Function Documentation

14.22.1.1 bml_trace()

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

Calculate trace of a matrix.

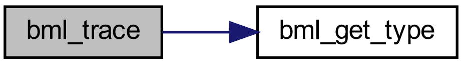
Parameters

A	Matrix tocalculate trace for

Returns

Trace of A

Here is the call graph for this function:



14.22.1.2 bml_trace_mult()

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

Calculate trace of a matrix multiplication.

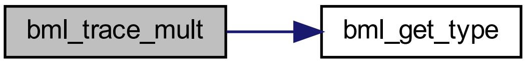
Parameters

A	Matrix A
B	Matrix B

Returns

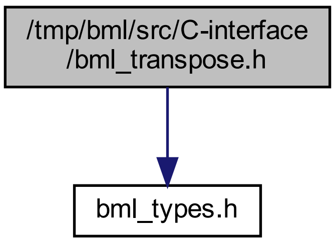
Trace of A∗B

Here is the call graph for this function:

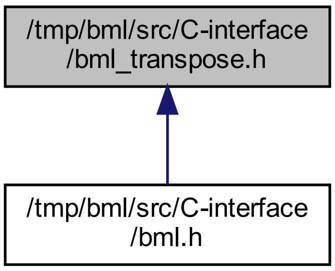


14.23 /tmp/bml/src/C-interface/bml_transpose.h File Reference

```
#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)

14.23.1 Function Documentation

14.23.1.1 bml_transpose()

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

Transpose matrix.

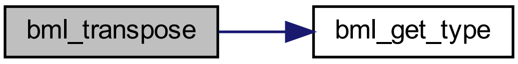
Parameters

A	Matrix to be transposed

Returns

Transposed A

Here is the call graph for this function:



14.23.1.2 bml_transpose_new()

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

Transpose matrix.

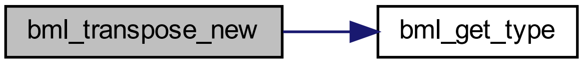
Parameters

A	Matrix to be transposed

Returns

Transposed A

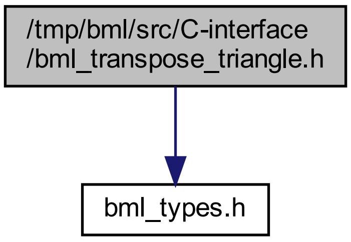
Here is the call graph for this function:



14.24 /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)

14.24.1 Function Documentation

14.24.1.1 bml_transpose_triangle()

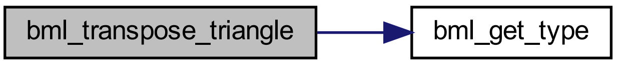
```
void bml_transpose_triangle (
            bml_matrix_t * A,
            char triangle )
```

Transposes a triangle of a matrix in place.

Parameters

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

Here is the call graph for this function:



14.25 /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 }
- 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 }

14.25.1 Typedef Documentation

14.25.1.1 bml_matrix_t

```
typedef void bml_matrix_t
```

The matrix type.

14.25.1.2 bml_vector_t

```
typedef void bml_vector_t
```

The vector type.

14.25.2 Enumeration Type Documentation

14.25.2.1 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.

14.25.2.2 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.

14.25.2.3 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).

14.25.2.4 bml_matrix_type_t

```
enum bml_matrix_type_t
```

The supported matrix types.

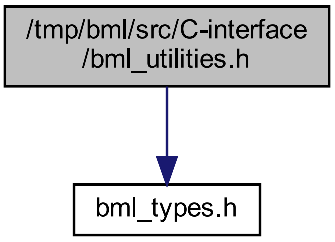
Enumerator

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

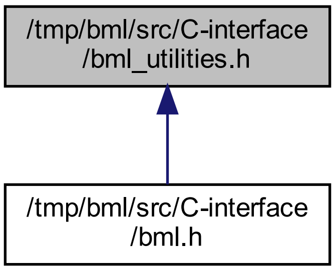
14.26 /tmp/bml/src/C-interface/bml_types_private.h File Reference

14.27 /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:



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)

14.27.1 Function Documentation

14.27.1.1 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.

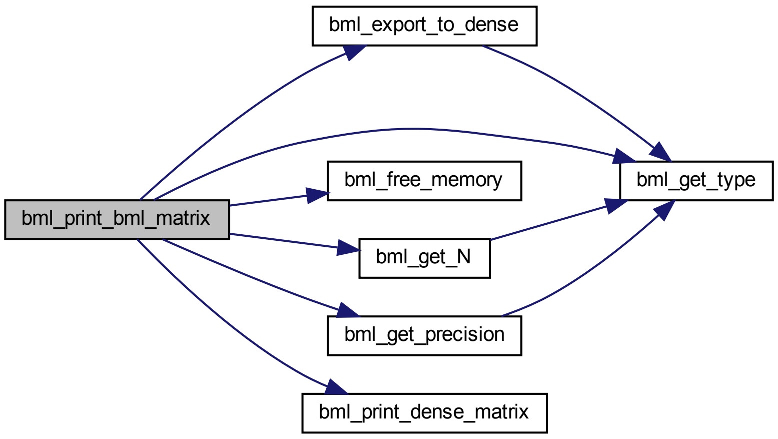
Parameters

A	The matrix.
i↩ _↩ *l*	The lower row index.
i↩ _↩ *u*	The upper row index.

Parameters

j←↩ _←↩ l	The lower column index.
j←↩ _←↩ u	The upper column index.

Here is the call graph for this function:



14.27.1.2 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.
i←↩ _←↩ l	The lower row index.
i←↩ _←↩ u	The upper row index.

14.27.1.3 bml_print_dense_matrix()

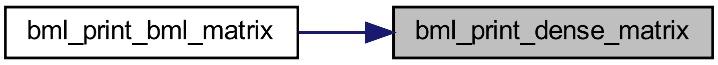
```
void bml_print_dense_matrix (
             int N,
             bml_matrix_precision_t matrix_precision,
             bml_dense_order_t order,
             void * A,
             int i_l,
             int i_u,
             int j_l,
             int j_u )
```

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:



14.27.1.4 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

N	The number of rows/columns.

Parameters

matrix_precision	The real precision.
v	The vector.
i_l	The lower row index.
i_u	The upper row index.

14.27.1.5 bml_read_bml_matrix()

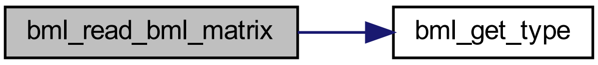
```
void bml_read_bml_matrix (
             bml_matrix_t * A,
             char * filename )
```

Read a bml matrix from a Matrix Market file.

Parameters

A	The matrix
filename	The file containing matrix

Here is the call graph for this function:



14.27.1.6 bml_write_bml_matrix()

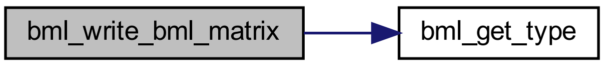
```
void bml_write_bml_matrix (
             bml_matrix_t * A,
             char * filename )
```

Write a bml matrix to a Matrix Market file.

Parameters

A	The matrix
filename	The file containing matrix

Here is the call graph for this function:



Index