ASAGI User Manual

0.5

Generated by Doxygen 1.8.7

Mon Oct 12 2015 15:58:59

Contents

1 Building and Installing ASAGI

1.1 Pre-requirements

1.1.1 Compiler

ASAGI requires at least GCC 4.7 or Intel Compiler 12.1

1.1.2 MPI

ASAGI makes use of the RMA (Remote Memory Access) API of the MPI-2 standard to transfer data. An MPI library that supports the new standard is required.

1.1.3 NetCDF

ASAGI uses the NetCDF library (`http://www.unidata.ucar.edu/software/netcdf/`) to load data files.

1.1.4 POSIX Threads (optional)

The PThreads library is required if ASAGI is compiled with NUMA support. If available it is also used to guaranty thread safety as an alternative to `std::mutex`.

1.1.5 NUMA policy library (optional)

ASASGI uses the NUMA library to detect NUMA domains. This library is required if ASAGI should be compiled with NUMA support.

1.2 Compilation

To generate the Makefiles, CMake is used. For CMake it is recommend to keep source and build directory apart:

```
mkdir build
cd build
cmake <path/to/asagi_sources>
```

Several environment variables affect the behavior of CMake. They must be set before running "cmake".

- **Compiler** The compiler can be selected by setting `CC` (C compiler), `CXX` (C++ compiler) and `FC` (Fortran compiler) environment variables. C and Fortran compiler are only required for C and Fortran examples and tests.

- **Libraries** The CMAKE_PREFIX_PATH is used when searching for the MPI, NetCDF, POSIX Threads and NUMA library. If NetCDF was configured with -prefix=<install_dir> for example, set CMAKE_P↩ REFIX_PATH=<install_dir>.

Besides the environment variables, you can change the behavior by setting internal CMake variables. They can be configured by adding one ore more -D<variable>=<value> options when running "cmake". These variables can also be changed later with the following command:

```
ccmake <path/to/asagi_build>
```

The important variables are listed below. Most of the variables are ASAGI specific and will not work with other CMake projects.

- **CMAKE_BUILD_TYPE = Debug** | **Release** When set to "Debug", additional run-time checks are enabled as well as debug messages. [Release]
- **CMAKE_INSTALL_PREFIX** Installation directory for ASAGI. [/usr/local/]
- **SHARED_LIB = ON** | **OFF** Build shared library. [ON]
- **STATIC_LIB = ON** | **OFF** Build static library. [OFF]
- **FORTRAN_SUPPORT = ON** | **OFF** Compile with Fortran support. [ON]
- **MAX_DIMENSIONS** Maximum number of dimensions supported by ASAGI [4]
- **THREADSAFE = ON** | **OFF** If enabled all ASAGI functions are thread-safe. This is required, for example, if ASAGI is used in hybrid MPI/OpenMP programs. [ON]
- **THREADSAFE_COUNTER = ON** | **OFF** Make access counters thread-safe. This may lead to a performance loss but makes sure, counters are accurate. [OFF]
- **THREADSAFE_MPI = ON** | **OFF** Make MPI calls thread-safe. This is required if the MPI library is not thread-safe by itself. [ON]
- **NOMPI = ON** | **OFF** Do not compile with MPI support. All algorithms that require MPI communication will be disabled. [OFF]
- **NONUMA = ON** | **OFF** Do not compile with NUMA support. All intra-node communications will be turned off. [OFF]
- **TESTS = ON** | **OFF** Compile tests. [OFF]
- **EXAMPLES = ON** | **OFF** Compile example programs. [OFF]

1.3 Tests

If you have enabled the tests, you can run them with the following command:

```
make test
```

1.4 Installation

To install ASAGI simply run:

```
make install
```

This will install the (static and/or shared) library as well as the header files. If pkg-config was found, this command will also install a pkg-config configuration file for ASAGI in CMAKE_INSTALL_PREFIX/lib/pkgconfig

You can install ASAGI with and without MPI support on your system. The version with MPI will be called asagi and the version without MPI asagi_nompi. Use the same include file for both libraries, but if you do not compile your program with MPI, make sure to define ASAGI_NOMPI before including the ASAGI header:

```
#define ASAGI_NOMPI
```

2 Using ASAGI

2.1 Minimal examples

These are minimal C, C++ and Fortran examples that load a 2-dimensional grid and print the value at (0,0). In each case the grid contains floating point values.

C example:

```c
#include <mpi.h>
#include <asagi.h>
#include <stdio.h>

int main(int argc, char** argv)
{
  MPI_Init(&argc, &argv);

  asagi_grid* grid = asagi_grid_create(ASAGI_FLOAT);
  asagi_grid_set_comm(grid, MPI_COMM_WORLD);
  // with threads, set number of threads
  asagi_grid_set_threads(grid, 1);

  if (asagi_grid_open(grid, "/path/to/netcdf/file.nc", 0) != ASAGI_SUCCESS) {
    printf("Could not load file\n");
    return 1;
  }

  double pos[] = {0, 0};
  printf("Value at (0,0): %f\n", asagi_grid_get_float(grid, pos, 0));

  asagi_grid_close(grid);

  MPI_Finalize();

  return 0;
}
```

C++ example:

```cpp
#include <mpi.h>
#include <asagi.h>
#include <iostream>

using namespace asagi;

int main(int argc, char** argv)
{
  MPI_Init(&argc, &argv);

  Grid* grid = Grid::create();
  grid->setComm(MPI_COMM_WORLD);
  // with threads, set number of threads
  grid->setThreads(1);

  if (grid->open("/path/to/netcdf/file.nc") != Grid::SUCCESS) {
    std::cout << "Could not load file" << std::endl;
    return 1;
  }

  double pos[] = {0, 0};
  std::cout << "Value at (0,0): " << grid->getFloat(pos) << std::endl;

  // The same as: "Grid::close(grid);"
  delete grid;

  MPI_Finalize();

  return 0;
}
```

Fortran example:

```
! You have two options:
! - Include the module file _once_ in your project:
```

```
!include 'asagi.f90'
! - Compile and link the module file as any other file in your project

program minimal
  use mpi
  use asagi
  use, intrinsic :: iso_c_binding
  implicit none

  integer :: grid_id
  real( kind=c_double ), dimension(2) :: pos
  integer :: error

  call mpi_init( error )

  grid_id = asagi_grid_create( )
  call asagi_grid_set_comm( grid_id, mpi_comm_world )
  ! with threads, set number of threads
  call asagi_grid_set_threads( grid_id, 1 );

  if( asagi_grid_open( grid_id, "/path/to/netcdf/file.nc" ) /= asagi_success ) then
    write (*,*) "Could not load file"
    call exit(1)
  end if

  pos(:) = 0
  write (*,*) "Value at (0,0):", asagi_grid_get_float( grid_id, pos )

  call asagi_grid_close( grid_id )

  call mpi_finalize( error )
end program minimal
```

2.2 Grid types

ASAGI distinguishes between three different grid types:

- **FULL** The whole grid will be loaded during the initialization. The file is not accessed during runtime. (default)
- **CACHE** ASAGI is used as a cache. After initialization, the cache will be empty. Each access to an element, will put the corresponding block into the cache for later usage.
- **PASS-THROUGH** ASAGI will pass each access to the underlying file system without any caching, etc.

Full storage does not automatically mean, that the full grid is stored on every CPU. If asagi::Grid::setComm() and/or asagi::Grid::setThreads() are called, the initial grid will be distributed among all nodes resp. CPUs. If the cache-grid is used and asagi::Grid::setThreads() and/or asagi::Grid::setComm() are set, ASAGI will copy the data from other NUMA domains and/or other MPI processes. Only if it is not available in another cache, the data will be fetched from the file.

2.3 Dimensions

ASAGI supports grids with up to `MAX_DIMENSIONS` dimensions. (`MAX_DIMENSIONS` is 4 by default, but can be changed during compilation of ASAGI.) The number of actual dimensions in the grid cannot be specified by calling an ASAGI function but depends on the netCDF input file.

Remarks

The order in the dimension in the netCDF file is in Fortran style (column-major, see NetCDF files) but the ASAGI interface uses C/C++ ordering (row-major).

2.4 Level of detail

A grid can have multiple resolutions. Each resolution is identified by a level id (level of detail). If the number of levels is not specified when creating a grid, the grid will contain only one level of detail. In this case you can also omit the level id in all other functions, since level 0 will be used by default. (C does not support default arguments or overloading, therefore omitting arguments is not possible when using the C interface.)

For grids with multiple levels asagi::Grid::open() must be called once for each level. Several levels can be stored in a single NetCDF file with different variable names. (Use asagi::Grid::setParam() to specify the variable name.) The coarsest resolution should have the level id 0. With ascending level id, the resolution should get finer. When accessing values with any `get` function, the level of detail can be selected with the last argument. The function asagi::Grid::close() has to be called only once for the whole grid.

2.5 Coordinate mapping

ASAGI distinguishes between actual coordinates and internal array indexes. All functions, that return a grid value, expect actual coordinates. ASAGI maps each coordinate to an array index using the coordinate variables from the NetCDF file (see section NetCDF files on how specify coordinate variables in NetCDF files). If no coordinate variable is available, the mapping is omitted. After the mapping, the coordinate is rounded to the nearest array index. ASAGI does not interpolate between array values.

The actual range of the grid can be obtained with asagi::Grid::getMin()/asagi::Grid::getMax(). They also return coordinates, not array indexes. It is erroneous to access values outside range of the grid.

The range of a dimension can be $(-\infty, \infty)$. This is the case if the size of the dimension in the netCDF file is one.

2.6 Value position

ASAGI supports cell-centered and vertex-centered grids. The value position can be switched with asagi::Grid::setParam().

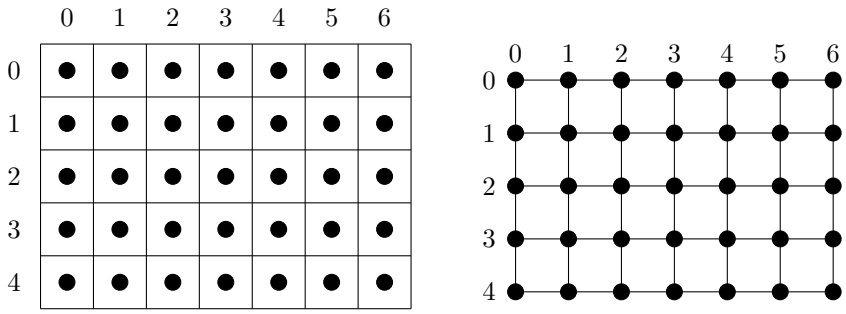

Figure 1: Cell-centered and vertex-centered grids

2.7 NetCDF files

All NetCDF files opened with ASAGI should respect the COARDS conventions (http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html). However, ASAGI has some further limitations:

- The attributes `scale_factor` and `add_offset` are ignored. Besides conversion between data types, ASAGI does not modify the values.
- Since ASAGI does not change the NetCDF file, all values have to be present in the file. Attributes, like `_FillValue` and `missing_value`, are not supported.
- ASAGI is not aware of any units. It is up to the user of the library to interpret the values correctly.
- Variables with more than three dimensions are not supported.

It is possible to open a NetCDF file by different grids or levels at the same time. This allows you, for example, to store all levels of one grid in a single NetCDF file. In this case the levels must be distinguished by the variable names.

2.8 Multi-thread support

When compiled with `THREADSAFE=ON` (see section Compilation) all functions are thread-safe. However, there are some restrictions due to MPI implementations. If your MPI library is not thread-safe, you have to add the additional flag `THREADSAFE_MPI=ON` which will mare sure that ASAGI does not call MPI functions from different threads at the same time. However, in this case, you are not allowed to call MPI **and** ASAGI functions at the same time.

Multi-thread support is required if you want to use ASAGI's NUMA functionality (see NUMA).

2.9 NUMA

ASAGI is able to detect the NUMA domains of your node. If more than one NUMA domain is detected, ASAGI will place a cache on each NUMA domain to increase locality. To enable the NUMA detection, call asagi::Grid::setThreads() with the **total** number of threads you are using. In this case, asagi::Grid::open() has to be called by all threads and is a collective operation.

2.10 Parameters

ASAGI supports several parameters for each grid:

Name	Values	Description	Grid-global (∗)
GRID	FULL | CACHE | PASS-THROUGH	The grid type (see Grid types)	yes
NUMA-CACHE	YES | NO	For full-grids, try local caches before using MPI	yes
MPI_COMMUNICATION	THREAD | WINDOW	Use a communication thread or MPI RMA (windows) for MPI communication (default: WINDOW, see MPI Communication)	yes
VALUE-POSITION	CELL-CENTERED | VERTEX-CENTERED	The value position (see Value position)	yes
TIME-DIMENSION	int	The dimension that holds the time (default is -1 which means no time dimension exists). ASAGI treats time dimension specially.	yes
VARIABLE	string	The variable name in the netCDf file. (default: z)	no
BLOCK-SIZE-*X*	int	The size of a block in dimension *X*. Use a negative value to set the block size equal to the total number cells in this dimension.	no

CACHE-SIZE	int	The size of the cache (in blocks) on each CPU.	no
CACHE-HAND-SPREAD	int	ASAGI uses the clock algorithm to approx. LRU. This parameter specifies the difference of the 2 hands in the clock. Lower values result in a faster algorithm but a worse approximation.	no

(∗) If yes, the parameter can only be set for all levels at the same time. Set the parameter `level` in asagi::Grid::setParam() to 0 to change value.

2.10.1 MPI Communication

ASAGI supports two different MPI communication patterns: Via MPI remote memory access (MPI windows) or a separate communication thread. The MPI windows are used by default since they do not have any special requirement and are easy to use. However, in some MPI libraries, RMA is poorly tested and does not work well, especially with hybrid parallelization.

Therefore, you can use the communication thread. In this mode, a separate thread is required which is responsible for answering remote requests. You have to start the thread with asagi::Grid::startCommThread **before** any grid using the communication thread is opened. Multiple grids will share one communication thread does you must not start more than communication. However, you have to make sure that the MPI communicator for the communication thread includes all grid communicators. Once the last grid using the communication thread is closed, you should stop the additional with asagi::Grid::stopCommThread. To use the communication thread it is also necessary to have a thread-safe MPI implementation.

2.11 Access counters

ASAGI supports several access counters to measure the throughput of the library and get information about effectiveness of the caches:

Name	Description
accesses	Total number of data accesses
numa_transfers	Number of blocks transfered between CPUs
mpi_transfers	Number of blocks transfered between processes
file_load	Number of blocks loaded from file (after initialization)
local_hits	Number values that where already in local NUMA domain
node_hits	Number values that where already on the local node
local_misses	Number of values that where not already in local memory

Remarks

If more than one thread is used and ASAGI is not compiled with `THREADSAFE_COUNTER=ON`, the counters might be inaccurate.

3 Troubleshooting

3.1 CMake does not find MPI

On some plattforms, CMake has problems finding MPI. Try to set the environment variable `CMAKE_PREFIX_PATH` (see section Building and Installing ASAGI) or select the MPI compiler before running CMake by setting the enviroment variable `CXX`.

3.2 The program hangs

Due to a bug (http://software.intel.com/en-us/forums/showthread.php?t=103456) in the Intel MPI library (version 4.0 update 3 and probably earlier versions) the remote memory access in ASAGI does not work properly. This only happens when fabric is set to "ofa" or "shm:ofa". Selecting a different fabric by changing the environment variable "I_MPI_FABRICS" solves the problem.

3.3 The program fails with "PMPI_Win_create: Assertion

'winptr->lock_table[i]' failed" or "function:MPI_WIN_LOCK, Invalid win argument"

The SGI Message Passing Toolkit uses a special mapped memory for one-sided communication. For large grids the default size of mapped memory may be too small. It is possible to increase the size by setting the environment variable `MPI_MAPPED_HEAP_SIZE`.

4 Module Documentation

4.1 Fortran Interface

Data Types

- module asagi

Functions/Subroutines

- integer(kind=c_int) function asagi::asagi_grid_create_struct (count, block_length, displacments, types)
- subroutine asagi::asagi_grid_set_comm (grid_id, comm)
- subroutine asagi::asagi_grid_set_threads (grid_id, threads)
- integer(kind=c_int) function asagi::asagi_grid_dimensions (grid_id)
- real(kind=c_double) function asagi::asagi_grid_min (grid_id, n)
- real(kind=c_double) function asagi::asagi_grid_max (grid_id, n)
- integer(kind=c_int) function asagi::asagi_grid_var_size (grid_id)
- subroutine asagi::asagi_grid_close (grid_id)
- integer(kind=c_int) function asagi::asagi_start_comm_thread (sched_cpu, comm)
- subroutine asagi::asagi_stop_comm_thread ()
- integer(kind=c_int) function asagi::asagi_node_local_rank (comm)
- integer function asagi::asagi_grid_create (type)
- integer function asagi::asagi_grid_create_array (basictype)
- subroutine asagi::asagi_grid_set_param (grid_id, name, value, level)
- integer function asagi::asagi_grid_open (grid_id, filename, level)
- real(kind=c_double) function asagi::asagi_grid_delta (grid_id, n, level)
- character function asagi::asagi_grid_get_byte (grid_id, pos, level)
- integer function asagi::asagi_grid_get_int (grid_id, pos, level)
- integer(kind=c_long) function asagi::asagi_grid_get_long (grid_id, pos, level)
- real function asagi::asagi_grid_get_float (grid_id, pos, level)
- real(kind=c_double) function asagi::asagi_grid_get_double (grid_id, pos, level)
- subroutine asagi::asagi_grid_get_buf (grid_id, buf, pos, level)

4.1.1 Detailed Description

4.1.2 Class Documentation

4.1.2.1 module asagi

ASAGI Fortran Interface.

Public Member Functions

- integer(kind=c_int) function asagi_grid_create_struct (count, block_length, displacments, types)
- subroutine asagi_grid_set_comm (grid_id, comm)
- subroutine asagi_grid_set_threads (grid_id, threads)
- integer(kind=c_int) function asagi_grid_dimensions (grid_id)
- real(kind=c_double) function asagi_grid_min (grid_id, n)
- real(kind=c_double) function asagi_grid_max (grid_id, n)
- integer(kind=c_int) function asagi_grid_var_size (grid_id)
- subroutine asagi_grid_close (grid_id)
- integer(kind=c_int) function asagi_start_comm_thread (sched_cpu, comm)
- subroutine asagi_stop_comm_thread ()
- integer(kind=c_int) function asagi_node_local_rank (comm)
- integer function asagi_grid_create (type)
- integer function asagi_grid_create_array (basictype)
- subroutine asagi_grid_set_param (grid_id, name, value, level)
- integer function asagi_grid_open (grid_id, filename, level)
- real(kind=c_double) function asagi_grid_delta (grid_id, n, level)
- character function asagi_grid_get_byte (grid_id, pos, level)
- integer function asagi_grid_get_int (grid_id, pos, level)
- integer(kind=c_long) function asagi_grid_get_long (grid_id, pos, level)
- real function asagi_grid_get_float (grid_id, pos, level)
- real(kind=c_double) function asagi_grid_get_double (grid_id, pos, level)
- subroutine asagi_grid_get_buf (grid_id, buf, pos, level)

4.1.3 Function/Subroutine Documentation

4.1.3.1 subroutine asagi::asagi_grid_close (integer(kind=c_int) *grid_id*)

See also

asagi::Grid::close(asagi::Grid∗)

4.1.3.2 integer function asagi::asagi_grid_create (integer, intent(in), optional *type*)

See also

asagi::Grid::create()

4.1.3.3 integer function asagi::asagi_grid_create_array (integer, intent(in), optional *basictype*)

See also

asagi::Grid::createArray()

4.1.3.4 integer(kind=c_int) function asagi::asagi_grid_create_struct (integer(kind=c_int) *count,* integer(kind=c_int), dimension(∗), intent(in) *block_length,* integer(kind=c_long), dimension(∗), intent(in) *displacments,* integer(kind=c_int), dimension(∗), intent(in) *types*)

See also

asagi::Grid::createStruct()

4.1.3.5 real(kind=c_double) function asagi::asagi_grid_delta (integer, intent(in) *grid_id,* integer, intent(in) *n,* integer, intent(in), optional *level*)

See also

asagi::Grid::getDelta()

4.1.3.6 integer(kind=c_int) function asagi::asagi_grid_dimensions (integer(kind=c_int) *grid_id*)

See also

agagi::Grid::getDimensions()

4.1.3.7 subroutine asagi::asagi_grid_get_buf (integer, intent(in) *grid_id,* type(c_ptr) *buf,* real(kind=c_double), dimension(∗), intent(in) *pos,* integer, intent(in), optional *level*)

See also

asagi::Grid::getBuf()

4.1.3.8 character function asagi::asagi_grid_get_byte (integer, intent(in) *grid_id,* real(kind=c_double), dimension(∗), intent(in) *pos,* integer, intent(in), optional *level*)

See also

asagi::Grid::getByte()

4.1.3.9 real(kind=c_double) function asagi::asagi_grid_get_double (integer, intent(in) *grid_id,* real(kind=c_double), dimension(∗), intent(in) *pos,* integer, intent(in), optional *level*)

See also

asagi::Grid::getDouble()

4.1.3.10 real function asagi::asagi_grid_get_float (integer, intent(in) *grid_id,* real(kind=c_double), dimension(∗), intent(in) *pos,* integer, intent(in), optional *level*)

See also

asagi::Grid::getFloat()

4.1.3.11 integer function asagi::asagi_grid_get_int (integer, intent(in) *grid_id,* real(kind=c_double), dimension(∗), intent(in) *pos,* integer, intent(in), optional *level*)

See also

asagi::Grid::getInt()

4.1.3.12 integer(kind=c_long) function asagi::asagi_grid_get_long (integer, intent(in) *grid_id,* real(kind=c_double), dimension(∗), intent(in) *pos,* integer, intent(in), optional *level*)

See also

asagi::Grid::getLong()

4.1.3.13 real(kind=c_double) function asagi::asagi_grid_max (integer(kind=c_int) *grid_id,* integer(kind=c_int) *n*)

See also

asagi::Grid::getMax()

4.1.3.14 real(kind=c_double) function asagi::asagi_grid_min (integer(kind=c_int) *grid_id,* integer(kind=c_int) *n*)

See also

asagi::Grid::getMin()

4.1.3.15 integer function asagi::asagi_grid_open (integer, intent(in) *grid_id,* character∗(∗), intent(in) *filename,* integer, intent(in), optional *level*)

See also

asagi::Grid::open()

4.1.3.16 subroutine asagi::asagi_grid_set_comm (integer(kind=c_int) *grid_id,* integer(kind=c_int) *comm*)

See also

asagi::Grid::setComm()

4.1.3.17 subroutine asagi::asagi_grid_set_param (integer, intent(in) *grid_id,* character∗(∗), intent(in) *name,* character∗(∗), intent(in) *value,* integer, intent(in), optional *level*)

See also

asagi::Grid::setParam()

4.1.3.18 subroutine asagi::asagi_grid_set_threads (integer(kind=c_int) *grid_id,* integer(kind=c_int) *threads*)

See also

asagi::Grid::setThreads()

4.1.3.19 integer(kind=c_int) function asagi::asagi_grid_var_size (integer(kind=c_int) *grid_id*)

See also

asagi::Grid::getVarSize()

4.1.3.20 integer(kind=c_int) function asagi::asagi_node_local_rank (integer(kind=c_int) *comm*)

See also

asagi::Grid::nodeLocalRank

4.1.3.21 integer(kind=c_int) function asagi::asagi_start_comm_thread (integer(kind=c_int) *sched_cpu,* integer(kind=c_int) *comm*)

See also

asagi::Grid::startCommThread

4.1.3.22 subroutine asagi::asagi_stop_comm_thread ( )

See also

asagi::Grid::stopCommThread

4.2 C Interface

Typedefs

- typedef asagi::Grid asagi_grid

Enumerations

- enum asagi_type
- enum asagi_error

Functions

- asagi_grid * asagi_grid_create (asagi_type type)
- asagi_grid * asagi_grid_create_array (asagi_type basic_type)
- asagi_grid * asagi_grid_create_struct (unsigned int count, unsigned int blockLength[], unsigned long displacements[], asagi_type types[])
- void asagi_grid_set_comm (asagi_grid *handle, MPI_Comm comm)
- void asagi_grid_set_threads (asagi_grid *handle, unsigned int threads)
- void asagi_grid_set_param (asagi_grid *handle, const char *name, const char *value, unsigned int level)
- asagi_error asagi_grid_open (asagi_grid *handle, const char *filename, unsigned int level)
- unsigned int asagi_grid_dimensions (asagi_grid *handle)
- double asagi_grid_min (asagi_grid *handle, unsigned int n)
- double asagi_grid_max (asagi_grid *handle, unsigned int n)
- double asagi_grid_delta (asagi_grid *handle, unsigned int n, unsigned int level)
- unsigned int asagi_grid_var_size (asagi_grid *handle)
- unsigned char asagi_grid_get_byte (asagi_grid *handle, const double *pos, unsigned int level)
- int asagi_grid_get_int (asagi_grid *handle, const double *pos, unsigned int level)
- long asagi_grid_get_long (asagi_grid *handle, const double *pos, unsigned int level)
- float asagi_grid_get_float (asagi_grid *handle, const double *pos, unsigned int level)
- double asagi_grid_get_double (asagi_grid *handle, const double *pos, unsigned int level)
- void asagi_grid_get_buf (asagi_grid *handle, void *buf, const double *pos, unsigned int level)
- void asagi_grid_close (asagi_grid *handle)
- asagi_error asagi_start_comm_thread (int sched_cpu, MPI_Comm comm)
- void asagi_stop_comm_thread ()
- int asagi_node_local_rank (MPI_Comm comm)

4.2.1 Detailed Description

4.2.2 Typedef Documentation

4.2.2.1 typedef struct asagi_grid asagi_grid

A handle for a grid

4.2.3 Enumeration Type Documentation

4.2.3.1 enum asagi_error

See also

asagi::Grid::Error

4.2.3.2 enum asagi_type

See also

asagi::Grid::Type

4.2.4 Function Documentation

4.2.4.1 void asagi_grid_close (asagi_grid * *handle*)

See also

asagi::Grid::close(asagi::Grid*)

4.2.4.2 asagi_grid* asagi_grid_create (asagi_type *type*)

See also

asagi::Grid::create()

4.2.4.3 asagi_grid* asagi_grid_create_array (asagi_type *basic_type*)

See also

asagi::Grid::createArray()

4.2.4.4 asagi_grid* asagi_grid_create_struct (unsigned int *count,* unsigned int *blockLength[],* unsigned long *displacements[],* asagi_type *types[]*)

See also

asagi::Grid::createStruct()

4.2.4.5 double asagi_grid_delta (asagi_grid * *handle,* unsigned int *n,* unsigned int *level*)

See also

asagi::Grid::getDelta()

4.2.4.6 unsigned int asagi_grid_dimensions (asagi_grid * *handle*)

See also

asagi::Grid::getDimensions()

4.2.4.7 void asagi_grid_get_buf (asagi_grid * *handle,* void * *buf,* const double * *pos,* unsigned int *level*)

See also

asagi::Grid::getBuf()

4.2.4.8 unsigned char asagi_grid_get_byte (asagi_grid * *handle,* const double * *pos,* unsigned int *level*)

See also

asagi::Grid::getByte()

4.2.4.9 double asagi_grid_get_double (asagi_grid * *handle,* const double * *pos,* unsigned int *level*)

See also

asagi::Grid::getDouble()

4.2.4.10 float asagi_grid_get_float (asagi_grid * *handle*, const double * *pos*, unsigned int *level*)

See also

asagi::Grid::getFloat()

4.2.4.11 int asagi_grid_get_int (asagi_grid * *handle*, const double * *pos*, unsigned int *level*)

See also

asagi::Grid::getInt()

4.2.4.12 long asagi_grid_get_long (asagi_grid * *handle*, const double * *pos*, unsigned int *level*)

See also

asagi::Grid::getLong()

4.2.4.13 double asagi_grid_max (asagi_grid * *handle*, unsigned int *n*)

See also

asagi::Grid::getMax()

4.2.4.14 double asagi_grid_min (asagi_grid * *handle*, unsigned int *n*)

See also

asagi::Grid::getMin()

4.2.4.15 asagi_error asagi_grid_open (asagi_grid * *handle*, const char * *filename*, unsigned int *level*)

See also

asagi::Grid::open()

4.2.4.16 void asagi_grid_set_comm (asagi_grid * *handle*, MPI_Comm *comm*)

See also

asagi::Grid::setComm()

4.2.4.17 void asagi_grid_set_param (asagi_grid * *handle*, const char * *name*, const char * *value*, unsigned int *level*)

See also

asagi::Grid::setParam()

4.2.4.18 void asagi_grid_set_threads (asagi_grid * *handle*, unsigned int *threads*)

See also

asagi::Grid::setThreads()

4.2.4.19 unsigned int asagi_grid_var_size (asagi_grid * *handle*)

See also

asagi::Grid::getVarSize()

4.2.4.20 int asagi_node_local_rank (MPI_Comm *comm*)

See also

asagi::Grid::nodeLocalRank(MPI_Comm)

4.2.4.21 asagi_error asagi_start_comm_thread (int *sched_cpu,* MPI_Comm *comm*)

See also

asagi::Grid::startCommThread(int, MPI_Comm)

4.2.4.22 void asagi_stop_comm_thread ( )

See also

asagi::Grid::stopCommThread()

4.3 C++ Interface

Classes

- class asagi::Grid

Enumerations

- enum asagi::Grid::Type {
  asagi::Grid::BYTE, asagi::Grid::INT, asagi::Grid::LONG, asagi::Grid::FLOAT,
  asagi::Grid::DOUBLE }
- enum asagi::Grid::Error {
  asagi::Grid::SUCCESS = 0, asagi::Grid::MPI_ERROR, asagi::Grid::THREAD_ERROR, asagi::Grid::NUMA↩
  _ERROR,
  asagi::Grid::UNKNOWN_PARAM, asagi::Grid::INVALID_VALUE, asagi::Grid::NOT_INITIALIZED, asagi::↩
  Grid::ALREADY_INITIALIZED,
  asagi::Grid::NOT_OPEN, asagi::Grid::VAR_NOT_FOUND, asagi::Grid::WRONG_SIZE, asagi::Grid::UNS↩
  UPPORTED_DIMENSIONS,
  asagi::Grid::INVALID_VAR_SIZE }

Functions

- virtual asagi::Grid::∼Grid ()
- virtual Error asagi::Grid::setComm (MPI_Comm comm=MPI_COMM_WORLD)=0
- virtual Error asagi::Grid::setThreads (unsigned int threads)=0
- virtual void asagi::Grid::setParam (const char ∗name, const char ∗value, unsigned int level=0)=0
- virtual Error asagi::Grid::open (const char ∗filename, unsigned int level=0)=0
- virtual unsigned int asagi::Grid::getDimensions () const =0
- virtual double asagi::Grid::getMin (unsigned int n) const =0
- virtual double asagi::Grid::getMax (unsigned int n) const =0
- virtual double asagi::Grid::getDelta (unsigned int n, unsigned int level=0) const =0
- virtual unsigned int asagi::Grid::getVarSize () const =0
- virtual unsigned char asagi::Grid::getByte (const double ∗pos, unsigned int level=0)=0
- virtual int asagi::Grid::getInt (const double ∗pos, unsigned int level=0)=0
- virtual long asagi::Grid::getLong (const double ∗pos, unsigned int level=0)=0
- virtual float asagi::Grid::getFloat (const double ∗pos, unsigned int level=0)=0
- virtual double asagi::Grid::getDouble (const double ∗pos, unsigned int level=0)=0
- virtual void asagi::Grid::getBuf (void ∗buf, const double ∗pos, unsigned int level=0)=0
- virtual unsigned long asagi::Grid::getCounter (const char ∗name, unsigned int level=0)=0
- static asagi::Grid ∗ asagi::Grid::create (Type type=FLOAT)
- static asagi::Grid ∗ asagi::Grid::createArray (Type type=FLOAT)
- static asagi::Grid ∗ asagi::Grid::createStruct (unsigned int count, unsigned int blockLength[], unsigned long displacements[], Type types[])
- static void asagi::Grid::close (asagi::Grid ∗grid)
- static Error asagi::Grid::startCommThread (int schedCPU=-1, MPI_Comm comm=MPI_COMM_WORLD)
- static void asagi::Grid::stopCommThread ()
- static int asagi::Grid::nodeLocalRank (MPI_Comm comm=MPI_COMM_WORLD)

4.3.1 Detailed Description

4.3.2 Class Documentation

4.3.2.1 class asagi::Grid

C++ Interface for ASAGI grids.

Public Types

- enum Type {
  BYTE, INT, LONG, FLOAT,
  DOUBLE }
- enum Error {
  SUCCESS = 0, MPI_ERROR, THREAD_ERROR, NUMA_ERROR,
  UNKNOWN_PARAM, INVALID_VALUE, NOT_INITIALIZED, ALREADY_INITIALIZED,
  NOT_OPEN, VAR_NOT_FOUND, WRONG_SIZE, UNSUPPORTED_DIMENSIONS,
  INVALID_VAR_SIZE }

Public Member Functions

- virtual ~Grid ()
- virtual Error setComm (MPI_Comm comm=MPI_COMM_WORLD)=0
- virtual Error setThreads (unsigned int threads)=0
- virtual void setParam (const char *name, const char *value, unsigned int level=0)=0
- virtual Error open (const char *filename, unsigned int level=0)=0
- virtual unsigned int getDimensions () const =0
- virtual double getMin (unsigned int n) const =0
- virtual double getMax (unsigned int n) const =0
- virtual double getDelta (unsigned int n, unsigned int level=0) const =0
- virtual unsigned int getVarSize () const =0
- virtual unsigned char getByte (const double *pos, unsigned int level=0)=0
- virtual int getInt (const double *pos, unsigned int level=0)=0
- virtual long getLong (const double *pos, unsigned int level=0)=0
- virtual float getFloat (const double *pos, unsigned int level=0)=0
- virtual double getDouble (const double *pos, unsigned int level=0)=0
- virtual void getBuf (void *buf, const double *pos, unsigned int level=0)=0
- virtual unsigned long getCounter (const char *name, unsigned int level=0)=0

Static Public Member Functions

- static asagi::Grid * create (Type type=FLOAT)
- static asagi::Grid * createArray (Type type=FLOAT)
- static asagi::Grid * createStruct (unsigned int count, unsigned int blockLength[], unsigned long displacements[], Type types[])
- static void close (asagi::Grid *grid)
- static Error startCommThread (int schedCPU=-1, MPI_Comm comm=MPI_COMM_WORLD)
- static void stopCommThread ()
- static int nodeLocalRank (MPI_Comm comm=MPI_COMM_WORLD)

4.3.3 Enumeration Type Documentation

4.3.3.1 enum asagi::Grid::Error

Possible errors that could occur

Enumerator

SUCCESS No error

MPI_ERROR An MPI function failed

THREAD_ERROR A pthread function failed

NUMA_ERROR An error in the NUMA detection code

UNKNOWN_PARAM Unknown configuration parameter

INVALID_VALUE Invalid configuration value

NOT_INITIALIZED Function is not yet initialized

ALREADY_INITIALIZED Function already initialized

NOT_OPEN Could not open input file

VAR_NOT_FOUND netCDF variable not found

WRONG_SIZE Wrong variable size in the file

UNSUPPORTED_DIMENSIONS Unsupported number of dimensions input file

INVALID_VAR_SIZE Variable size in the input file does not match the type

4.3.3.2 enum asagi::Grid::Type

The primitive data types supported by ASAGI

Enumerator

BYTE signed byte

INT signed 4-byte integer

LONG signed 8-byte integer

FLOAT 4-byte floating point value

DOUBLE 8-byte floating point value

4.3.4 Function Documentation

4.3.4.1 static void asagi::Grid::close (asagi::Grid * *grid*) [inline], [static]

Frees all memory resources associated with `grid`. After a grid is closed you cannot access any values and you can not reopen another NetCDF file.
This function does the same as calling `delete grid;` and it is the C++ equivalent to asagi_grid_close(asagi_grid*) and asagi::asagi_grid_close

Parameters

grid	The grid that should be closed.

4.3.4.2 static asagi::Grid* asagi::Grid::create (Type *type* = FLOAT) [static]

Creates a new grid containing values with a primitive data type

Parameters

type	The type of the values in the grid

4.3.4.3 static asagi::Grid* asagi::Grid::createArray (Type *type* = FLOAT) [static]

Creates a new grid containing arrays

The length of the arrays is determined by the input file

Parameters

type	The type of the values in the arrays

4.3.4.4 static asagi::Grid* asagi::Grid::createStruct (unsigned int *count*, unsigned int *blockLength[]*, unsigned long *displacements[]*, Type *types[]*) [static]

Creates a new grid containing structured values

Parameters

count	Number of blocks in the structure
blockLength	Number of elements in each block
displacements	Displacement of each block
types	Primitive types of the blocks

4.3.4.5 virtual void asagi::Grid::getBuf (void * *buf*, const double * *pos*, unsigned int *level* = 0) [pure virtual]

Copys the element at `pos` into buf. The buffer size has to be (at least) getVarSize() bytes.

Parameters

buf	Pointer to the buffer where the data should be written
pos	The coordinates of the value, the array must have at least the size of the dimension of the grid
level	The level from which the data should be fetched

4.3.4.6 virtual unsigned char asagi::Grid::getByte (const double * *pos*, unsigned int *level* = 0) [pure virtual]

If the grid contains array values, only the first element of the array is returned

Parameters

pos	The coordinates of the value, the array must have at least the size of the dimension of the grid
level	The level from which the data should be fetched

Returns

The element at `pos` as a char

4.3.4.7 virtual unsigned long asagi::Grid::getCounter (const char * *name*, unsigned int *level* = 0) [pure virtual]

Gets the current value of a counter for a grid level.

See Access counters for a list of all counters.

Returns

The current counter value or 0 if the name is not defined

Warning

The performance counters are by default not thread-safe for performance reason. You may get wrong result when using more than one thread.

4.3.4.8 virtual double asagi::Grid::getDelta (unsigned int *n*, unsigned int *level* = 0) const [pure virtual]

Parameters

n	The dimension
level	The level for which the difference is requested

Returns

The difference of two coordinates in dimension `n`

4.3.4.9 virtual unsigned int asagi::Grid::getDimensions ( ) const [pure virtual]

Returns the number of dimensions loaded from the file.

Returns

The number of dimensions of the grid

4.3.4.10 virtual double asagi::Grid::getDouble (const double * *pos*, unsigned int *level* = 0) [pure virtual]

Parameters

pos	The coordinates of the value, the array must have at least the size of the dimension of the grid
level	The level from which the data should be fetched

Returns

The element at `pos` as a double

See also

getByte

4.3.4.11 virtual float asagi::Grid::getFloat (const double * *pos*, unsigned int *level* = 0) [pure virtual]

Parameters

pos	The coordinates of the value, the array must have at least the size of the dimension of the grid
level	The level from which the data should be fetched

Returns

The element at `pos` as a float

See also

getByte

4.3.4.12 virtual int asagi::Grid::getInt (const double * *pos*, unsigned int *level* = 0) [pure virtual]

Parameters

pos	The coordinates of the value, the array must have at least the size of the dimension of the grid
level	The level from which the data should be fetched

Returns

The element at `pos` as an integer

See also

getByte

4.3.4.13 virtual long asagi::Grid::getLong (const double * *pos*, unsigned int *level* = 0) [pure virtual]

Parameters

pos	The coordinates of the value, the array must have at least the size of the dimension of the grid
level	The level from which the data should be fetched

Returns

The element at `pos` as a long

See also

getByte

4.3.4.14 virtual double asagi::Grid::getMax (unsigned int *n*) const [pure virtual]

Parameters

n	The dimension

Returns

The maximum allowed coordinate in dimension `n`

4.3.4.15 virtual double asagi::Grid::getMin (unsigned int *n*) const [pure virtual]

Parameters

n	The dimension

Returns

The minimum allowed coordinate in dimension `n`

4.3.4.16 virtual unsigned int asagi::Grid::getVarSize ( ) const [pure virtual]

Returns

The number of bytes that are stored in each grid cell

4.3.4.17 static int asagi::Grid::nodeLocalRank (MPI_Comm *comm* = `MPI_COMM_WORLD`) [static]

Computes the rank of the process on the node. This function can be used to determine the core, to which the communication threads should be pinned in startCommThread.

Parameters

comm	The communicator that should be used

Returns

The node local rank

4.3.4.18 virtual Error asagi::Grid::open (const char * *filename*, unsigned int *level* = `0`) [pure virtual]

Loads values from a NetCDF file.

This function must be called for each level of detail. If more than one thread is used, this is a collective function for all threads.

4.3.4.19 virtual Error asagi::Grid::setComm (MPI_Comm *comm* = MPI_COMM_WORLD) [pure virtual]

Call this function before open() if the grids should exchange chunks via MPI.

4.3.4.20 virtual void asagi::Grid::setParam (const char * *name*, const char * *value*, unsigned int *level* = 0) [pure virtual]

Changes a grid parameter.

This function allows you to change ASAGI's configuration. It must be called before calling open(const char*, unsigned int).

See Parameters for a list of supported parameters.

Parameters

name	The name of the parameter
value	The new value for the parameter
level	Change the parameter for the specified level of detail. Should be 0 when setting **value-position**

4.3.4.21 virtual Error asagi::Grid::setThreads (unsigned int *threads*) [pure virtual]

Sets the number of threads in the application.

This function must be called before open(). If it is not called, one thread is assumed.

4.3.4.22 static Error asagi::Grid::startCommThread (int *schedCPU* = -1, MPI_Comm *comm* = MPI_COMM_WORLD) [static]

Starts the communication thread. This must be done before a grid with the option MPI_COMMUNICATION = THREAD is opened.

This is a collective operation within comm.

Parameters

schedCPU	The id of the CPU on which the communication thread should run. Use negative values to select the last, second last, ... CPU. If the id is invalid, the thread will not be pinned to a CPU.
comm	The communicator specifying which processes will be involved in any communication

Returns

SUCCESS or an error

4.3.4.23 static void asagi::Grid::stopCommThread ( ) [static]

Stops the communication thread

4.3.4.24 virtual asagi::Grid::~Grid ( ) [inline], [virtual]

See also

close(asagi::Grid*)

Index