

**schwarz-lib**

Generated automatically from doc-setup

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Installation Instructions</b>	<b>3</b>
<b>3</b>	<b>Testing Instructions</b>	<b>5</b>
<b>4</b>	<b>Benchmarking.</b>	<b>7</b>
<b>5</b>	<b>Module Documentation</b>	<b>9</b>
5.1	Communicate . . . . .	9
5.1.1	Detailed Description . . . . .	9
5.2	Initialization . . . . .	10
5.2.1	Detailed Description . . . . .	10
5.3	Schwarz Class . . . . .	11
5.3.1	Detailed Description . . . . .	11
5.4	Solve . . . . .	12
5.4.1	Detailed Description . . . . .	12
5.5	Utils . . . . .	13
5.5.1	Detailed Description . . . . .	13
<b>6</b>	<b>Namespace Documentation</b>	<b>15</b>
6.1	ProcessTopology Namespace Reference . . . . .	15
6.1.1	Detailed Description . . . . .	15
6.2	SchwarzWrappers Namespace Reference . . . . .	15
6.2.1	Detailed Description . . . . .	16
6.3	SchwarzWrappers::CommHelpers Namespace Reference . . . . .	16
6.3.1	Detailed Description . . . . .	16
6.4	SchwarzWrappers::ConvergenceTools Namespace Reference . . . . .	16
6.4.1	Detailed Description . . . . .	16
6.5	SchwarzWrappers::PartitionTools Namespace Reference . . . . .	17
6.5.1	Detailed Description . . . . .	17
6.6	SchwarzWrappers::SolverTools Namespace Reference . . . . .	17
6.6.1	Detailed Description . . . . .	17

<b>7</b>	<b>Class Documentation</b>	<b>19</b>
7.1	BadDimension Class Reference . . . . .	19
7.1.1	Detailed Description . . . . .	19
7.1.2	Constructor & Destructor Documentation . . . . .	19
7.1.2.1	BadDimension() . . . . .	19
7.2	SchwarzWrappers::Settings::comm_settings Struct Reference . . . . .	20
7.2.1	Detailed Description . . . . .	21
7.3	SchwarzWrappers::Communicate< ValueType, IndexType >::comm_struct Struct Reference . . . . .	21
7.3.1	Detailed Description . . . . .	22
7.3.2	Member Data Documentation . . . . .	22
7.3.2.1	global_get . . . . .	22
7.3.2.2	global_put . . . . .	23
7.3.2.3	is_local_neighbor . . . . .	23
7.3.2.4	local_get . . . . .	23
7.3.2.5	local_put . . . . .	24
7.3.2.6	remote_get . . . . .	24
7.3.2.7	remote_put . . . . .	24
7.4	SchwarzWrappers::Communicate< ValueType, IndexType > Class Template Reference . . . . .	25
7.4.1	Detailed Description . . . . .	25
7.4.2	Member Function Documentation . . . . .	26
7.4.2.1	exchange_boundary() . . . . .	26
7.4.2.2	local_to_global_vector() . . . . .	26
7.4.2.3	setup_windows() . . . . .	27
7.4.2.4	update_boundary() . . . . .	27
7.5	SchwarzWrappers::Settings::convergence_settings Struct Reference . . . . .	28
7.5.1	Detailed Description . . . . .	28
7.6	CudaError Class Reference . . . . .	28
7.6.1	Detailed Description . . . . .	29
7.6.2	Constructor & Destructor Documentation . . . . .	29
7.6.2.1	CudaError() . . . . .	29

7.7	CusparsError Class Reference	29
7.7.1	Detailed Description	30
7.7.2	Constructor & Destructor Documentation	30
7.7.2.1	CusparsError()	30
7.8	SchwarzWrappers::device_guard Class Reference	30
7.8.1	Detailed Description	30
7.9	SchwarzWrappers::Initialize< ValueType, IndexType > Class Template Reference	31
7.9.1	Detailed Description	31
7.9.2	Member Function Documentation	32
7.9.2.1	generate_rhs()	32
7.9.2.2	partition()	32
7.9.2.3	setup_global_matrix_laplacian()	33
7.9.2.4	setup_local_matrices()	34
7.9.2.5	setup_vectors()	35
7.10	SchwarzWrappers::Metadata< ValueType, IndexType > Struct Template Reference	36
7.10.1	Detailed Description	37
7.10.2	Member Data Documentation	38
7.10.2.1	local_solver_tolerance	38
7.10.2.2	tolerance	38
7.11	MetisError Class Reference	38
7.11.1	Detailed Description	38
7.11.2	Constructor & Destructor Documentation	38
7.11.2.1	MetisError()	38
7.12	SchwarzWrappers::SchwarzBase< ValueType, IndexType > Class Template Reference	39
7.12.1	Detailed Description	40
7.12.2	Constructor & Destructor Documentation	40
7.12.2.1	SchwarzBase()	40
7.12.3	Member Function Documentation	42
7.12.3.1	print_matrix()	42
7.12.3.2	print_vector()	43

7.12.3.3	<code>run()</code>	43
7.13	<code>SchwarzWrappers::Settings</code> Struct Reference	45
7.13.1	Detailed Description	46
7.13.2	Member Data Documentation	46
7.13.2.1	<code>explicit_laplacian</code>	46
7.13.2.2	<code>naturally_ordered_factor</code>	47
7.14	<code>SchwarzWrappers::Solve&lt; ValueType, IndexType &gt;</code> Class Template Reference	47
7.14.1	Detailed Description	47
7.15	<code>SchwarzWrappers::SolverRAS&lt; ValueType, IndexType &gt;</code> Class Template Reference	47
7.15.1	Detailed Description	48
7.15.2	Constructor & Destructor Documentation	48
7.15.2.1	<code>SolverRAS()</code>	48
7.15.3	Member Function Documentation	49
7.15.3.1	<code>exchange_boundary()</code>	49
7.15.3.2	<code>setup_local_matrices()</code>	50
7.15.3.3	<code>setup_windows()</code>	53
7.15.3.4	<code>update_boundary()</code>	55
7.16	<code>SchwarzWrappers::Utils&lt; ValueType, IndexType &gt;</code> Struct Template Reference	56
7.16.1	Detailed Description	56
<b>Index</b>		<b>59</b>

# Chapter 1

## Main Page

This is the main page for the Schwarz library pdf documentation. The repository is hosted on [github](#). Documentation on aspects such as the build system, can be found at the [Installation Instructions](#) page.

### Modules

The structure of the Schwarz Library code is divided into different [modules](#) :

- [Initialization](#) : Handles the initialization of the problem and the solver.
- [Communicate](#) : Handles the communication.
- [Solve](#) : Handles the local solution and the convergence detection.
- [Schwarz Class](#) : The Classes related to the Schwarz solvers.
- [Utils](#) : Provides some basic utilities.





## Chapter 2

# Installation Instructions

### Building

Use the standard cmake build procedure:

```
mkdir build; cd build
cmake -G "Unix Makefiles" [OPTIONS] .. && make
```

Replace [OPTIONS] with desired cmake options for your build. The library adds the following additional switches to control what is being built:

- `-DSCHWARZ_BUILD_BENCHMARKING={ON, OFF}` Builds some example benchmarks. Default is ON
- `-DSCHWARZ_BUILD_METIS={ON, OFF}` Builds with support for the METIS partitioner. User needs to provide the path to the installation of the METIS library in `METIS_DIR`, preferably as an environment variable. Default is OFF
- `-DSCHWARZ_BUILD_CHOLMOD={ON, OFF}` Builds with support for the CHOLMOD module from the Suitesparse library. User needs to set an environment variable `CHOLMOD_DIR` to the path containing the CHOLMOD installation. Default is OFF
- `-DSCHWARZ_BUILD_CUDA={ON, OFF}` Builds with CUDA support. Though Ginkgo provides most of the required CUDA support, we do need to link to CUDA for explicit setting of GPU affinities, some custom gather and scatter operations. Default is OFF.
- `-DSCHWARZ_BUILD_CLANG_TIDY={ON, OFF}` Builds with support for clang-tidy Default is OFF
- `-DSCHWARZ_BUILD_DEALII={ON, OFF}` Builds with support for the finite element library deal.ii Default is OFF
- `-DSCHWARZ_WITH_HWLOC={ON, OFF}` Builds with support for the hardware locality library used for binding hardware. hwloc is distributed as a part of the Open-MPI project. Default is ON
- `-DSCHWARZ_DEVEL_TOOLS={ON, OFF}` Builds with some developer tools support. Default is ON. In particular uses `git-cmake-format` to automatically format the source files with `clang-format`.

### Tips

- If you are having CUDA problems and you are not using CUDA, then feel free to switch the CUDA module off with `-DSCHWARZ_BUILD_CUDA=off`.
- Installing CHOLMOD can be a bit annoying. TODO add some details on fixing Suitesparse compilation.
- When doing merge commits it is possible that make format does not work. You can run `cmake -DSCHWARZ_DEVEL_TOOLS=OFF ..` to temporarily switch off the formatting. Please switch it on again when committing normally.



## **Chapter 3**

# **Testing Instructions**

Will be updated soon.



## Chapter 4

# Benchmarking.

### Benchmark example 1.

**Poisson solver using Restricted Additive Schwarz with overlap.**

The flag `-DSCHWARZ_BUILD_BENCHMARKING` (default ON) enables the example and benchmarking snippets. The following command line options are available for this example. This is setup using `gflags`.

The executable is run in the following fashion:

```
“sh [MPI_COMMAND] [MPI_OPTIONS]
```



## Chapter 5

# Module Documentation

### 5.1 Communicate

A module dedicated to the Communication interface in schwarz-lib.

#### Namespaces

- [SchwarzWrappers::CommHelpers](#)  
*The `CommHelper` namespace .*
- [ProcessTopology](#)  
*The `ProcessTopology` namespace .*

#### Classes

- class [SchwarzWrappers::Communicate< ValueType, IndexType >](#)  
*The communication class that provides the methods for the communication between the subdomains.*
- struct [SchwarzWrappers::Metadata< ValueType, IndexType >](#)  
*The solver metadata struct.*

#### 5.1.1 Detailed Description

A module dedicated to the Communication interface in schwarz-lib.

## 5.2 Initialization

A module dedicated to the initialization and setup and usage of the solvers in schwarz-lib.

### Namespaces

- [SchwarzWrappers::PartitionTools](#)  
*The [PartitionTools](#) namespace .*
- [ProcessTopology](#)  
*The [ProcessTopology](#) namespace .*

### Classes

- class [SchwarzWrappers::device\\_guard](#)  
*This class defines a device guard for the cuda functions and the cuda module.*
- class [SchwarzWrappers::Initialize< ValueType, IndexType >](#)  
*The initialization class that provides methods for initialization of the solver.*
- struct [SchwarzWrappers::Settings](#)  
*The struct that contains the solver settings and the parameters to be set by the user.*
- struct [SchwarzWrappers::Metadata< ValueType, IndexType >](#)  
*The solver metadata struct.*

### 5.2.1 Detailed Description

A module dedicated to the initialization and setup and usage of the solvers in schwarz-lib.



## 5.3 Schwarz Class

A module dedicated to the Schwarz solver classes in schwarz-lib.

### Classes

- class [SchwarzWrappers::SolverRAS< ValueType, IndexType >](#)  
*An implementation of the solver interface using the RAS solver.*
- class [SchwarzWrappers::SchwarzBase< ValueType, IndexType >](#)  
*The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.*

### 5.3.1 Detailed Description

A module dedicated to the Schwarz solver classes in schwarz-lib.

## 5.4 Solve

A module dedicated to the solvers including local solution and convergence detection in schwarz-lib.

### Namespaces

- [SchwarzWrappers::ConvergenceTools](#)  
*The [ConvergenceTools](#) namespace .*
- [SchwarzWrappers::SolverTools](#)  
*The [SolverTools](#) namespace .*

### Classes

- struct [SchwarzWrappers::Metadata](#)< [ValueType](#), [IndexType](#) >  
*The solver metadata struct.*
- class [SchwarzWrappers::Solve](#)< [ValueType](#), [IndexType](#) >  
*The Solver class the provides the solver and the convergence checking methods.*

#### 5.4.1 Detailed Description

A module dedicated to the solvers including local solution and convergence detection in schwarz-lib.

## 5.5 Utils

A module dedicated to the utilities in schwarz-lib.

### Classes

- struct [SchwarzWrappers::Utils< ValueType, IndexType >](#)  
*The utilities class which provides some checks and basic utilities.*

### 5.5.1 Detailed Description

A module dedicated to the utilities in schwarz-lib.



## Chapter 6

# Namespace Documentation

### 6.1 ProcessTopology Namespace Reference

The [ProcessTopology](#) namespace .

#### 6.1.1 Detailed Description

The [ProcessTopology](#) namespace .

proc\_topo

### 6.2 SchwarzWrappers Namespace Reference

The Schwarz wrappers namespace.

#### Namespaces

- [CommHelpers](#)

*The CommHelper namespace .*

- [ConvergenceTools](#)

*The ConvergenceTools namespace .*

- [PartitionTools](#)

*The PartitionTools namespace .*

- [SolverTools](#)

*The SolverTools namespace .*

## Classes

- class [Communicate](#)  
*The communication class that provides the methods for the communication between the subdomains.*
- class [device\\_guard](#)  
*This class defines a device guard for the cuda functions and the cuda module.*
- class [Initialize](#)  
*The initialization class that provides methods for initialization of the solver.*
- struct [Metadata](#)  
*The solver metadata struct.*
- class [SchwarzBase](#)  
*The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.*
- struct [Settings](#)  
*The struct that contains the solver settings and the parameters to be set by the user.*
- class [Solve](#)  
*The Solver class the provides the solver and the convergence checking methods.*
- class [SolverRAS](#)  
*An implementation of the solver interface using the RAS solver.*
- struct [Utils](#)  
*The utilities class which provides some checks and basic utilities.*

### 6.2.1 Detailed Description

The Schwarz wrappers namespace.

## 6.3 SchwarzWrappers::CommHelpers Namespace Reference

The CommHelper namespace .

### 6.3.1 Detailed Description

The CommHelper namespace .

comm\_helpers

## 6.4 SchwarzWrappers::ConvergenceTools Namespace Reference

The [ConvergenceTools](#) namespace .

### 6.4.1 Detailed Description

The [ConvergenceTools](#) namespace .

conv\_tools

## 6.5 SchwarzWrappers::PartitionTools Namespace Reference

The [PartitionTools](#) namespace .

### 6.5.1 Detailed Description

The [PartitionTools](#) namespace .

part\_tools

## 6.6 SchwarzWrappers::SolverTools Namespace Reference

The [SolverTools](#) namespace .

### 6.6.1 Detailed Description

The [SolverTools](#) namespace .

solver\_tools





## Chapter 7

# Class Documentation

### 7.1 BadDimension Class Reference

[BadDimension](#) is thrown if an operation is being applied to a LinOp with bad dimensions.

```
#include <exception.hpp>
```

#### Public Member Functions

- [BadDimension](#) (const std::string &file, int line, const std::string &func, const std::string &op\_name, std::size\_t op\_num\_rows, std::size\_t op\_num\_cols, const std::string &clarification)  
*Initializes a bad dimension error.*

#### 7.1.1 Detailed Description

[BadDimension](#) is thrown if an operation is being applied to a LinOp with bad dimensions.

#### 7.1.2 Constructor & Destructor Documentation

##### 7.1.2.1 BadDimension()

```
BadDimension::BadDimension (
    const std::string & file,
    int line,
    const std::string & func,
    const std::string & op_name,
    std::size_t op_num_rows,
    std::size_t op_num_cols,
    const std::string & clarification ) [inline]
```

Initializes a bad dimension error.

## Parameters

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The function name where the error occurred
<i>op_name</i>	The name of the operator
<i>op_num_rows</i>	The row dimension of the operator
<i>op_num_cols</i>	The column dimension of the operator
<i>clarification</i>	An additional message further describing the error

```

115         : Error(file, line,
116               func + ": Object " + op_name + " has dimensions [" +
117                   std::to_string(op_num_rows) + " x " +
118                   std::to_string(op_num_cols) + "]: " + clarification)
119     {}

```

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

- exception.hpp (7ee6ebb)

## 7.2 SchwarzWrappers::Settings::comm\_settings Struct Reference

The settings for the various available communication paradigms.

```
#include <settings.hpp>
```

### Public Attributes

- bool `enable_onesided` = false  
*Enable one-sided communication.*
- bool `enable_overlap` = false  
*Enable explicit overlap between communication and computation.*
- bool `enable_put` = false  
*Put the data to the window using MPI\_Put rather than get.*
- bool `enable_get` = true  
*Get the data to the window using MPI\_Get rather than put.*
- bool `enable_one_by_one` = false  
*Push each element separately directly into the buffer.*
- bool `enable_flush_local` = false  
*Use local flush.*
- bool `enable_flush_all` = true  
*Use flush all.*
- bool `enable_lock_local` = false  
*Use local locks.*
- bool `enable_lock_all` = true  
*Use lock all.*

### 7.2.1 Detailed Description

The settings for the various available communication paradigms.

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

- settings.hpp (7ee6ebb)

## 7.3 SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct Struct Reference

The communication struct used to store the communication data.

```
#include <communicate.hpp>
```

### Public Attributes

- int [num\\_neighbors\\_in](#)  
*The number of neighbors this subdomain has to receive data from.*
- int [num\\_neighbors\\_out](#)  
*The number of neighbors this subdomain has to send data to.*
- std::shared\_ptr< gko::Array< IndexType > > [neighbors\\_in](#)  
*The neighbors this subdomain has to receive data from.*
- std::shared\_ptr< gko::Array< IndexType > > [neighbors\\_out](#)  
*The neighbors this subdomain has to send data to.*
- std::vector< bool > [is\\_local\\_neighbor](#)  
*The bool vector which is true if the neighbors of a subdomain are in one node.*
- int [local\\_num\\_neighbors\\_in](#)  
*The number of neighbors this subdomain has to receive data from.*
- int [local\\_num\\_neighbors\\_out](#)  
*The number of neighbors this subdomain has to send data to.*
- std::shared\_ptr< gko::Array< IndexType > > [local\\_neighbors\\_in](#)  
*The neighbors this subdomain has to receive data from.*
- std::shared\_ptr< gko::Array< IndexType > > [local\\_neighbors\\_out](#)  
*The neighbors this subdomain has to send data to.*
- std::shared\_ptr< gko::Array< IndexType \* > > [global\\_put](#)  
*The array containing the number of elements that each subdomain sends from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [local\\_put](#)  
*The array containing the number of elements that each subdomain sends from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [remote\\_put](#)  
*The array containing the number of elements that each subdomain sends from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [global\\_get](#)  
*The array containing the number of elements that each subdomain gets from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [local\\_get](#)  
*The array containing the number of elements that each subdomain gets from the other.*
- std::shared\_ptr< gko::Array< IndexType \* > > [remote\\_get](#)  
*The array containing the number of elements that each subdomain gets from the other.*
- std::shared\_ptr< gko::Array< IndexType > > [window\\_ids](#)

- The RDMA window ids.*

  - `std::shared_ptr< gko::Array< IndexType > >` [windows\\_from](#)

*The RDMA window ids to receive data from.*

  - `std::shared_ptr< gko::Array< IndexType > >` [windows\\_to](#)

*The RDMA window ids to send data to.*

  - `std::shared_ptr< gko::Array< MPI_Request > >` [put\\_request](#)

*The put request array.*

  - `std::shared_ptr< gko::Array< MPI_Request > >` [get\\_request](#)

*The get request array.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [send\\_buffer](#)

*The send buffer used for the actual communication for both one-sided and two-sided.*

  - `std::shared_ptr< gko::matrix::Dense< ValueType > >` [recv\\_buffer](#)

*The recv buffer used for the actual communication for both one-sided and two-sided.*

  - `std::shared_ptr< gko::Array< IndexType > >` [get\\_displacements](#)

*The displacements for the receiving of the buffer.*

  - `std::shared_ptr< gko::Array< IndexType > >` [put\\_displacements](#)

*The displacements for the sending of the buffer.*

  - `MPI_Win` [window\\_recv\\_buffer](#)

*The RDMA window for the recv buffer.*

  - `MPI_Win` [window\\_send\\_buffer](#)

*The RDMA window for the send buffer.*

  - `MPI_Win` [window\\_x](#)

*The RDMA window for the solution vector.*

### 7.3.1 Detailed Description

```
template<typename ValueType, typename IndexType>
struct SchwarzWrappers::Communicate< ValueType, IndexType >::comm_struct
```

The communication struct used to store the communication data.

### 7.3.2 Member Data Documentation

#### 7.3.2.1 global\_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > SchwarzWrappers::Communicate< ValueType, IndexType
>::comm_struct::global_get
```

The array containing the number of elements that each subdomain gets from the other.

For example. `global_get[p][0]` contains the overall number of elements to be received to subdomain `p` and `global_←_put[p][i]` contains the index of the solution vector to be received from subdomain `p`.

Referenced by `SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize()`, `SchwarzWrappers::←SchwarzBase< ValueType, IndexType >::SchwarzBase()`, `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.2 global\_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > SchwarzWrappers::Communicate< ValueType, IndexType
>::comm_struct::global_put
```

The array containing the number of elements that each subdomain sends from the other.

For example. `global_put[p][0]` contains the overall number of elements to be sent to subdomain `p` and `global_put[p][i]` contains the index of the solution vector to be sent to subdomain `p`.

Referenced by `SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize()`, `SchwarzWrappers::SchwarzBase< ValueType, IndexType >::SchwarzBase()`, `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.3 is\_local\_neighbor

```
template<typename ValueType , typename IndexType >
std::vector<bool> SchwarzWrappers::Communicate< ValueType, IndexType >::comm_struct::is_
local_neighbor
```

The bool vector which is true if the neighbors of a subdomain are in one node.

Referenced by `SchwarzWrappers::SchwarzBase< ValueType, IndexType >::SchwarzBase()`, `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.4 local\_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > SchwarzWrappers::Communicate< ValueType, IndexType
>::comm_struct::local_get
```

The array containing the number of elements that each subdomain gets from the other.

For example. `global_get[p][0]` contains the overall number of elements to be received to subdomain `p` and `global_put[p][i]` contains the index of the solution vector to be received from subdomain `p`.

Referenced by `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.5 local\_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > SchwarzWrappers::Communicate< ValueType, IndexType
>::comm_struct::local_put
```

The array containing the number of elements that each subdomain sends from the other.

For example. `global_put[p][0]` contains the overall number of elements to be sent to subdomain `p` and `global_put[p][i]` contains the index of the solution vector to be sent to subdomain `p`.

Referenced by `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.6 remote\_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > SchwarzWrappers::Communicate< ValueType, IndexType
>::comm_struct::remote_get
```

The array containing the number of elements that each subdomain gets from the other.

For example. `global_get[p][0]` contains the overall number of elements to be received to subdomain `p` and `global_get[p][i]` contains the index of the solution vector to be received from subdomain `p`.

Referenced by `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.7 remote\_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > SchwarzWrappers::Communicate< ValueType, IndexType
>::comm_struct::remote_put
```

The array containing the number of elements that each subdomain sends from the other.

For example. `global_put[p][0]` contains the overall number of elements to be sent to subdomain `p` and `global_put[p][i]` contains the index of the solution vector to be sent to subdomain `p`.

Referenced by `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows()`.

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

- `communicate.hpp` (7ee6ebb)

## 7.4 SchwarzWrappers::Communicate< ValueType, IndexType > Class Template Reference

The communication class that provides the methods for the communication between the subdomains.

```
#include <communicate.hpp>
```

### Classes

- struct [comm\\_struct](#)

*The communication struct used to store the communication data.*

### Public Member Functions

- virtual void [setup\\_comm\\_buffers](#) ()=0  
*Sets up the communication buffers needed for the boundary exchange.*
- virtual void [setup\\_windows](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &main\_buffer)=0  
*Sets up the windows needed for the asynchronous communication.*
- virtual void [exchange\\_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &solution\_vector)=0  
*Exchanges the elements of the solution vector.*
- void [local\\_to\\_global\\_vector](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, const std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_vector, std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_vector)  
*Transforms data from a local vector to a global vector.*
- virtual void [update\\_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_solution, const std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_rhs, const std::shared\_ptr< gko::matrix::Dense< ValueType >> &solution\_vector, std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_old\_solution, const std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &interface\_matrix)=0  
*Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.*
- void [clear](#) ([Settings](#) &settings)  
*Clears the data.*

#### 7.4.1 Detailed Description

```
template<typename ValueType, typename IndexType>
class SchwarzWrappers::Communicate< ValueType, IndexType >
```

The communication class that provides the methods for the communication between the subdomains.

#### Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

#### [Communicate](#)

## 7.4.2 Member Function Documentation

### 7.4.2.1 exchange\_boundary()

```
template<typename ValueType , typename IndexType >
void SchwarzWrappers::Communicate< ValueType, IndexType >::exchange_boundary (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & solution_vector ) [pure
virtual]
```

Exchanges the elements of the solution vector.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>solution_vector</i>	The solution vector being exchanged between the subdomains.

Implemented in [SchwarzWrappers::SolverRAS< ValueType, IndexType >](#).

Referenced by [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::run\(\)](#).

### 7.4.2.2 local\_to\_global\_vector()

```
template<typename ValueType , typename IndexType >
void SchwarzWrappers::Communicate< ValueType, IndexType >::local_to_global_vector (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & local_vector,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_vector )
```

Transforms data from a local vector to a global vector.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>local_vector</i>	The local vector in question.
<i>global_vector</i>	The global vector in question.

```
70 {
71     using vec = gko::matrix::Dense<ValueType>;
72     auto alpha = gko::initialize<gko::matrix::Dense<ValueType>>(
73         {1.0}, settings.executor);
74     auto temp_vector = vec::create(
75         settings.executor, gko::dim<2>(metadata.local_size, 1),
```



```

76         (gko::Array<ValueType>::view(
77             settings.executor, metadata.local_size,
78             &global_vector->get_values()[metadata.first_row
79                 ->get_data()[metadata.my_rank]])),
80         1);
81
82     auto temp_vector2 = vec::create(
83         settings.executor, gko::dim<2>(metadata.local_size, 1),
84         (gko::Array<ValueType>::view(settings.executor, metadata.local_size,
85             &local_vector->get_values()[0])),
86         1);
87     if (settings.convergence_settings.convergence_crit ==
88         Settings::convergence_settings::local_convergence_crit::
89             residual_based) {
90         local_vector->add_scaled(alpha.get(), temp_vector.get());
91         temp_vector->add_scaled(alpha.get(), local_vector.get());
92     } else {
93         // TODO GPU: DONE
94         temp_vector->copy_from(temp_vector2.get());
95     }
96 }

```

### 7.4.2.3 setup\_windows()

```

template<typename ValueType , typename IndexType >
void SchwarzWrappers::Communicate< ValueType, IndexType >::setup_windows (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & main_buffer ) [pure virtual]

```

Sets up the windows needed for the asynchronous communication.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>main_buffer</i>	The main buffer being exchanged between the subdomains.

Implemented in [SchwarzWrappers::SolverRAS< ValueType, IndexType >](#).

Referenced by [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::run\(\)](#).

### 7.4.2.4 update\_boundary()

```

template<typename ValueType , typename IndexType >
void SchwarzWrappers::Communicate< ValueType, IndexType >::update_boundary (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & local_solution,
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & local_rhs,
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & solution_vector,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_old_solution,
    const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_↔
matrix ) [pure virtual]

```

Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.

**Parameters**

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>local_solution</i>	The local solution vector in the subdomain.
<i>local_rhs</i>	The local right hand side vector in the subdomain.
<i>solution_vector</i>	The workspace solution vector.
<i>global_old_solution</i>	The global solution vector of the previous iteration.
<i>interface_matrix</i>	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.

Implemented in [SchwarzWrappers::SolverRAS< ValueType, IndexType >](#).

Referenced by [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::run\(\)](#).

The documentation for this class was generated from the following files:

- [communicate.hpp \(7ee6ebb\)](#)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/communicate.cpp \(7ee6ebb\)](#)

## 7.5 SchwarzWrappers::Settings::convergence\_settings Struct Reference

The various convergence settings available.

```
#include <settings.hpp>
```

### 7.5.1 Detailed Description

The various convergence settings available.

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

- [settings.hpp \(7ee6ebb\)](#)

## 7.6 CudaError Class Reference

[CudaError](#) is thrown when a CUDA routine throws a non-zero error code.

```
#include <exception.hpp>
```

**Public Member Functions**

- [CudaError](#) (const std::string &file, int line, const std::string &func, int error\_code)  
*Initializes a CUDA error.*

### 7.6.1 Detailed Description

[CudaError](#) is thrown when a CUDA routine throws a non-zero error code.

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 CudaError()

```
CudaError::CudaError (
    const std::string & file,
    int line,
    const std::string & func,
    int error_code ) [inline]
```

Initializes a CUDA error.

#### Parameters

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The name of the CUDA routine that failed
<i>error_code</i>	The resulting CUDA error code

```
137         : Error(file, line, func + ": " + get_error(error_code))
138     {}
```

The documentation for this class was generated from the following files:

- exception.hpp (7ee6ebb)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (7ee6ebb)

## 7.7 CusparsedError Class Reference

[CusparsedError](#) is thrown when a cuSPARSE routine throws a non-zero error code.

```
#include <exception.hpp>
```

### Public Member Functions

- [CusparsedError](#) (const std::string &file, int line, const std::string &func, int error\_code)  
*Initializes a cuSPARSE error.*

### 7.7.1 Detailed Description

[CusparsedError](#) is thrown when a cuSPARSE routine throws a non-zero error code.

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 CusparsedError()

```
CusparsedError::CusparsedError (
    const std::string & file,
    int line,
    const std::string & func,
    int error_code ) [inline]
```

Initializes a cuSPARSE error.

#### Parameters

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The name of the cuSPARSE routine that failed
<i>error_code</i>	The resulting cuSPARSE error code

```
159         : Error(file, line, func + ": " + get_error(error_code))
160     {}
```

The documentation for this class was generated from the following files:

- exception.hpp (7ee6ebb)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (7ee6ebb)

## 7.8 SchwarzWrappers::device\_guard Class Reference

This class defines a device guard for the cuda functions and the cuda module.

```
#include <device_guard.hpp>
```

### 7.8.1 Detailed Description

This class defines a device guard for the cuda functions and the cuda module.

The guard is used to make sure that the device code is run on the correct cuda device, when run with multiple devices. The class records the current device id and uses `cudaSetDevice` to set the device id to the one being passed in. After the scope has been exited, the destructor sets the `device_id` back to the one before entering the scope.

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

- device\_guard.hpp (7ee6ebb)

## 7.9 SchwarzWrappers::Initialize< ValueType, IndexType > Class Template Reference

The initialization class that provides methods for initialization of the solver.

```
#include <initialization.hpp>
```

### Public Member Functions

- void [generate\\_rhs](#) (std::vector< ValueType > &rhs)  
*Generates the right hand side vector.*
- void [setup\\_global\\_matrix\\_laplacian](#) (const gko::size\_type &oned\_laplacian\_size, std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &global\_matrix)  
*Generates the 2D global laplacian matrix.*
- void [partition](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, const std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &global\_matrix, std::vector< unsigned int > &partition\_indices)  
*The partitioning function.*
- void [setup\\_vectors](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::vector< ValueType > &rhs, std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_rhs, std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_rhs, std::shared\_ptr< gko::matrix::Dense< ValueType >> &local\_solution, std::shared\_ptr< gko::matrix::Dense< ValueType >> &global\_solution)  
*Setup the vectors with default values and allocate memory if not allocated.*
- virtual void [setup\\_local\\_matrices](#) ([Settings](#) &settings, [Metadata](#)< ValueType, IndexType > &metadata, std::vector< unsigned int > &partition\_indices, std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &global\_matrix, std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &local\_matrix, std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &interface\_matrix, std::shared\_ptr< gko::matrix::Permutation< IndexType >> &local\_perm, std::shared\_ptr< gko::matrix::Permutation< IndexType >> &local\_inv\_perm)=0  
*Sets up the local and the interface matrices from the global matrix and the partition indices.*

### Public Attributes

- std::vector< unsigned int > [partition\\_indices](#)  
*The partition indices containing the subdomains to which each row(vertex) of the matrix(graph) belongs to.*
- std::vector< unsigned int > [cell\\_weights](#)  
*The cell weights for the partition algorithm.*

### Additional Inherited Members

#### 7.9.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class SchwarzWrappers::Initialize< ValueType, IndexType >
```

The initialization class that provides methods for initialization of the solver.

#### Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

## Initialization

### 7.9.2 Member Function Documentation

#### 7.9.2.1 generate\_rhs()

```
template<typename ValueType , typename IndexType >
void SchwarzWrappers::Initialize< ValueType, IndexType >::generate_rhs (
    std::vector< ValueType > & rhs )
```

Generates the right hand side vector.

##### Parameters

<i>rhs</i>	The rhs vector.
------------	-----------------

Referenced by SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize().

```
79 {
80     std::uniform_real_distribution<double> unif(0.0, 1.0);
81     std::default_random_engine engine;
82     for (gko::size_type i = 0; i < rhs.size(); ++i) {
83         rhs[i] = unif(engine);
84     }
85 }
```

#### 7.9.2.2 partition()

```
template<typename ValueType , typename IndexType >
void SchwarzWrappers::Initialize< ValueType, IndexType >::partition (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_↵
matrix,
    std::vector< unsigned int > & partition_indices )
```

The partitioning function.

Allows the partition of the global matrix depending with METIS and a regular 1D decomposition.

##### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>global_matrix</i>	The global matrix.
<i>partition_indices</i>	The partition indices [OUTPUT].

References SchwarzWrappers::Metadata< ValueType, IndexType >::global\_size, SchwarzWrappers::Metadata< ValueType, IndexType >::my\_rank, SchwarzWrappers::Metadata< ValueType, IndexType >::num\_subdomains, and SchwarzWrappers::Settings::write\_debug\_out.

Referenced by SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize().

```

263 {
264     partition_indices.resize(metadata.global_size);
265     if (metadata.my_rank == 0) {
266         auto partition_settings =
267             (Settings::partition_settings::partition_zoltan |
268              Settings::partition_settings::partitionmetis |
269              Settings::partition_settings::partition_regular |
270              Settings::partition_settings::partition_regular2d |
271              Settings::partition_settings::partition_custom) &
272             settings.partition;
273
274         if (partition_settings ==
275             Settings::partition_settings::partition_zoltan) {
276             SCHWARZ_NOT_IMPLEMENTED;
277         } else if (partition_settings ==
278             Settings::partition_settings::partitionmetis) {
279             if (metadata.my_rank == 0) {
280                 std::cout << " METIS partition" << std::endl;
281             }
282             PartitionTools::PartitionMetis(
283                 settings, global_matrix, this->cell_weights,
284                 metadata.num_subdomains, partition_indices);
285         } else if (partition_settings ==
286             Settings::partition_settings::partition_regular) {
287             if (metadata.my_rank == 0) {
288                 std::cout << " Regular 1D partition" << std::endl;
289             }
290             PartitionTools::PartitionRegular(
291                 global_matrix, metadata.num_subdomains, partition_indices);
292         } else if (partition_settings ==
293             Settings::partition_settings::partition_regular2d) {
294             if (metadata.my_rank == 0) {
295                 std::cout << " Regular 2D partition" << std::endl;
296             }
297             PartitionTools::PartitionRegular2D(
298                 global_matrix, settings.write_debug_out,
299                 metadata.num_subdomains, partition_indices);
300         } else if (partition_settings ==
301             Settings::partition_settings::partition_custom) {
302             // User partitions mesh manually
303             SCHWARZ_NOT_IMPLEMENTED;
304         } else {
305             SCHWARZ_NOT_IMPLEMENTED;
306         }
307     }
308 }

```

### 7.9.2.3 setup\_global\_matrix\_laplacian()

```

template<typename ValueType , typename IndexType >
void SchwarzWrappers::Initialize< ValueType, IndexType >::setup_global_matrix_laplacian (
    const gko::size_type & oned_laplacian_size,
    std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_matrix )

```

Generates the 2D global laplacian matrix.

#### Parameters

<i>oned_laplacian_size</i>	The size of the one d laplacian grid.
<i>global_matrix</i>	The global matrix.

Referenced by `SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize()`.

```

201 {
202     using index_type = IndexType;
203     using value_type = ValueType;
204     using mtx = gko::matrix::Csr<value_type, index_type>;
205     gko::size_type global_size = oned_laplacian_size *
        oned_laplacian_size;
206
207     global_matrix = mtx::create(settings.executor->get_master(),
208                                gko::dim<2>(global_size), 5 * global_size);
209     value_type *values = global_matrix->get_values();
210     index_type *row_ptrs = global_matrix->get_row_ptrs();
211     index_type *col_idxs = global_matrix->get_col_idxs();
212
213     std::vector<gko::size_type> exclusion_set;
214
215     std::map<IndexType, ValueType> stencil_map = {
216         {-oned_laplacian_size, -1}, {-1, -1}, {0, 4}, {1, -1},
217         {oned_laplacian_size, -1},
218     };
219     for (auto i = 2; i < global_size; ++i) {
220         gko::size_type index = (i - 1) * oned_laplacian_size;
221         if (index * index < global_size * global_size) {
222             exclusion_set.push_back(
223                 linearize_index(index, index - 1, global_size));
224             exclusion_set.push_back(
225                 linearize_index(index - 1, index, global_size));
226         }
227     }
228
229     std::sort(exclusion_set.begin(),
230              exclusion_set.begin() + exclusion_set.size());
231
232     IndexType pos = 0;
233     IndexType col_idx = 0;
234     row_ptrs[0] = pos;
235     gko::size_type cur_idx = 0;
236     for (IndexType i = 0; i < global_size; ++i) {
237         for (auto ofs : stencil_map) {
238             auto in_exclusion_flag =
239                 (exclusion_set[cur_idx] ==
240                  linearize_index(i, i + ofs.first, global_size));
241             if (0 <= i + ofs.first && i + ofs.first < global_size &&
242                 !in_exclusion_flag) {
243                 values[pos] = ofs.second;
244                 col_idxs[pos] = i + ofs.first;
245                 ++pos;
246             }
247             if (in_exclusion_flag) {
248                 cur_idx++;
249             }
250             col_idx = row_ptrs[i + 1] - pos;
251         }
252         row_ptrs[i + 1] = pos;
253     }
254 }

```

#### 7.9.2.4 setup\_local\_matrices()

```

template<typename ValueType, typename IndexType>
void SchwarzWrappers::Initialize< ValueType, IndexType >::setup_local_matrices (
    Settings & settings,
    Metadata< ValueType, IndexType > & metadata,
    std::vector< unsigned int > & partition_indices,
    std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_matrix,
    std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & local_matrix,
    std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_matrix,
    std::shared_ptr< gko::matrix::Permutation< IndexType >> & local_perm,
    std::shared_ptr< gko::matrix::Permutation< IndexType >> & local_inv_perm ) [pure
virtual]

```

Sets up the local and the interface matrices from the global matrix and the partition indices.



## Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>partition_indices</i>	The array containing the partition indices.
<i>global_matrix</i>	The global system matrix.
<i>local_matrix</i>	The local system matrix.
<i>interface_matrix</i>	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.
<i>local_perm</i>	The local permutation, obtained through RCM or METIS.

Implemented in [SchwarzWrappers::SolverRAS< ValueType, IndexType >](#).

Referenced by [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize\(\)](#).

## 7.9.2.5 setup\_vectors()

```
template<typename ValueType , typename IndexType >
void SchwarzWrappers::Initialize< ValueType, IndexType >::setup_vectors (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::vector< ValueType > & rhs,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & local_rhs,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_rhs,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & local_solution,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution )
```

Setup the vectors with default values and allocate mameory if not allocated.

## Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>local_rhs</i>	The local right hand side vector in the subdomain.
<i>global_rhs</i>	The global right hand side vector.
<i>local_solution</i>	The local solution vector in the subdomain.
<i>global_solution</i>	The global solution vector.

References [SchwarzWrappers::Settings::executor](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::first\\_row](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::global\\_size](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::local\\_size\\_x](#), and [SchwarzWrappers::Metadata< ValueType, IndexType >::my\\_rank](#).

Referenced by [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize\(\)](#).

```
319 {
320     using vec = gko::matrix::Dense<ValueType>;
321     auto my_rank = metadata.my_rank;
322     auto first_row = metadata.first_row->get_data()[my_rank];
323
324     // Copy the global rhs vector to the required executor.
325     gko::Array<ValueType> temp_rhs(settings.executor->get_master(), rhs.begin(),
```

```

326                                     rhs.end());
327     global_rhs = vec::create(settings.executor,
328                             gko::dim<2>(metadata.global_size, 1), temp_rhs, 1);
329     global_solution = vec::create(settings.executor->get_master(),
330                                  gko::dim<2>(metadata.global_size, 1));
331
332     local_rhs =
333         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
334     // Extract the local rhs from the global rhs. Also takes into account the
335     // overlap.
336     SolverTools::extract_local_vector(settings, metadata, local_rhs, global_rhs,
337                                       first_row);
338
339     local_solution =
340         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
341 }

```

The documentation for this class was generated from the following files:

- initialization.hpp (7ee6ebb)
- /home/runner/work/schwarz-lib/schwarz-lib/source/initialization.cpp (7ee6ebb)

## 7.10 SchwarzWrappers::Metadata< ValueType, IndexType > Struct Template Reference

The solver metadata struct.

```
#include <settings.hpp>
```

### Public Attributes

- MPI\_Comm [mpi\\_communicator](#)  
*The MPI communicator.*
- gko::size\_type [global\\_size](#) = 0  
*The size of the global matrix.*
- gko::size\_type [oned\\_laplacian\\_size](#) = 0  
*The size of the 1 dimensional laplacian grid.*
- gko::size\_type [local\\_size](#) = 0  
*The size of the local subdomain matrix.*
- gko::size\_type [local\\_size\\_x](#) = 0  
*The size of the local subdomain matrix + the overlap.*
- gko::size\_type [local\\_size\\_o](#) = 0  
*The size of the local subdomain matrix + the overlap.*
- gko::size\_type [overlap\\_size](#) = 0  
*The size of the overlap between the subdomains.*
- gko::size\_type [num\\_subdomains](#) = 1  
*The number of subdomains used within the solver.*
- int [my\\_rank](#)  
*The rank of the subdomain.*
- int [my\\_local\\_rank](#)  
*The local rank of the subdomain.*
- int [local\\_num\\_procs](#)  
*The local number of procs in the subdomain.*
- int [comm\\_size](#)  
*The number of subdomains used within the solver, size of the communicator.*

- int [num\\_threads](#)  
*The number of threads used within the solver for each subdomain.*
- IndexType [iter\\_count](#)  
*The iteration count of the solver.*
- ValueType [tolerance](#)  
*The tolerance of the complete solver.*
- ValueType [local\\_solver\\_tolerance](#)  
*The tolerance of the local solver in case of an iterative solve.*
- IndexType [max\\_iters](#)  
*The maximum iteration count of the solver.*
- unsigned int [precond\\_max\\_block\\_size](#)  
*The maximum block size for the preconditioner.*
- ValueType [current\\_residual\\_norm](#) = -1.0  
*The current residual norm of the subdomain.*
- ValueType [min\\_residual\\_norm](#) = -1.0  
*The minimum residual norm of the subdomain.*
- std::vector< std::tuple< int, int, int, std::string, std::vector< ValueType > > > [time\\_struct](#)  
*The struct used to measure the timings of each function within the solver loop.*
- std::vector< std::tuple< int, std::vector< std::tuple< int, int > >, std::vector< std::tuple< int, int > >, int, int > > [comm\\_data\\_struct](#)  
*The struct used to measure the timings of each function within the solver loop.*
- std::shared\_ptr< gko::Array< IndexType > > [global\\_to\\_local](#)  
*The mapping containing the global to local indices.*
- std::shared\_ptr< gko::Array< IndexType > > [local\\_to\\_global](#)  
*The mapping containing the local to global indices.*
- std::shared\_ptr< gko::Array< IndexType > > [overlap\\_row](#)  
*The overlap row indices.*
- std::shared\_ptr< gko::Array< IndexType > > [first\\_row](#)  
*The starting row of each subdomain in the matrix.*
- std::shared\_ptr< gko::Array< IndexType > > [permutation](#)  
*The permutation used for the re-ordering.*
- std::shared\_ptr< gko::Array< IndexType > > [i\\_permutation](#)  
*The inverse permutation used for the re-ordering.*

### 7.10.1 Detailed Description

```
template<typename ValueType, typename IndexType>
struct SchwarzWrappers::Metadata< ValueType, IndexType >
```

The solver metadata struct.

#### Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

## 7.10.2 Member Data Documentation

### 7.10.2.1 local\_solver\_tolerance

```
template<typename ValueType, typename IndexType>
ValueType SchwarzWrappers::Metadata< ValueType, IndexType >::local_solver_tolerance
```

The tolerance of the local solver in case of an iterative solve.

The residual norm reduction required.

### 7.10.2.2 tolerance

```
template<typename ValueType, typename IndexType>
ValueType SchwarzWrappers::Metadata< ValueType, IndexType >::tolerance
```

The tolerance of the complete solver.

The residual norm reduction required.

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

- settings.hpp (7ee6ebb)

## 7.11 MetisError Class Reference

[MetisError](#) is thrown when a METIS routine throws a non-zero error code.

```
#include <exception.hpp>
```

### Public Member Functions

- [MetisError](#) (const std::string &file, int line, const std::string &func, int error\_code)  
*Initializes a METIS error.*

### 7.11.1 Detailed Description

[MetisError](#) is thrown when a METIS routine throws a non-zero error code.

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 MetisError()

```
MetisError::MetisError (
    const std::string & file,
    int line,
    const std::string & func,
    int error_code ) [inline]
```

Initializes a METIS error.

## Parameters

<i>file</i>	The name of the offending source file
<i>line</i>	The source code line number where the error occurred
<i>func</i>	The name of the METIS routine that failed
<i>error_code</i>	The resulting METIS error code

```

182         : Error(file, line, func + ": " + get_error(error_code))
183     {}

```

The documentation for this class was generated from the following files:

- exception.hpp (7ee6ebb)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (7ee6ebb)

## 7.12 SchwarzWrappers::SchwarzBase< ValueType, IndexType > Class Template Reference

The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.

```
#include <schwarz_base.hpp>
```

### Public Member Functions

- [SchwarzBase](#) ([Settings](#) &settings, [Metadata](#)< ValueType, IndexType > &metadata)  
*The constructor that takes in the user settings and a metadata struct containing the solver metadata.*
- void [initialize](#) ()  
*Initialize the matrix and vectors.*
- void [run](#) (std::shared\_ptr< gko::matrix::Dense< ValueType >> &solution)  
*The function that runs the actual solver and obtains the final solution.*
- void [print\\_vector](#) (const std::shared\_ptr< gko::matrix::Dense< ValueType >> &vector, int subd, std::string name)  
*The auxiliary function that prints a passed in vector.*
- void [print\\_matrix](#) (const std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &matrix, int rank, std::string name)  
*The auxiliary function that prints a passed in CSR matrix.*

## Public Attributes

- `std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > >` [local\\_matrix](#)  
*The local subdomain matrix.*
- `std::shared_ptr< gko::matrix::Permutation< IndexType > >` [local\\_perm](#)  
*The local subdomain permutation matrix/array.*
- `std::shared_ptr< gko::matrix::Permutation< IndexType > >` [local\\_inv\\_perm](#)  
*The local subdomain inverse permutation matrix/array.*
- `std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > >` [triangular\\_factor](#)  
*The local triangular factor used for the triangular solves.*
- `std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > >` [interface\\_matrix](#)  
*The local interface matrix.*
- `std::shared_ptr< gko::matrix::Csr< ValueType, IndexType > >` [global\\_matrix](#)  
*The global matrix.*
- `std::shared_ptr< gko::matrix::Dense< ValueType > >` [local\\_rhs](#)  
*The local right hand side.*
- `std::shared_ptr< gko::matrix::Dense< ValueType > >` [global\\_rhs](#)  
*The global right hand side.*
- `std::shared_ptr< gko::matrix::Dense< ValueType > >` [local\\_solution](#)  
*The local solution vector.*
- `std::shared_ptr< gko::matrix::Dense< ValueType > >` [global\\_solution](#)  
*The global solution vector.*

## Additional Inherited Members

### 7.12.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class SchwarzWrappers::SchwarzBase< ValueType, IndexType >
```

The Base solver class is meant to be the class implementing the common implementations for all the schwarz methods.

It derives from the Initialization class, the Communication class and the [Solve](#) class all of which are templated.

#### Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 SchwarzBase()

```
template<typename ValueType , typename IndexType >
SchwarzWrappers::SchwarzBase< ValueType, IndexType >::SchwarzBase (
```

```

    Settings & settings,
    Metadata< ValueType, IndexType > & metadata )

```

The constructor that takes in the user settings and a metadata struct containing the solver metadata.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.

References SchwarzWrappers::Metadata< ValueType, IndexType >::comm\_size, SchwarzWrappers::Settings< cuda\_device\_guard, SchwarzWrappers::Settings::executor, SchwarzWrappers::Settings::executor\_string, SchwarzWrappers::Metadata< ValueType, IndexType >::first\_row, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::get\_displacements, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::global\_get, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::global\_put, SchwarzWrappers::Metadata< ValueType, IndexType >::global\_size, SchwarzWrappers::Metadata< ValueType, IndexType >::global\_to\_local, SchwarzWrappers::Metadata< ValueType, IndexType >::i\_permutation, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::is\_local\_neighbor, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_neighbors\_in, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_neighbors\_out, SchwarzWrappers::Metadata< ValueType, IndexType >::local\_num\_procs, SchwarzWrappers::Metadata< ValueType, IndexType >::local\_to\_global, SchwarzWrappers::Metadata< ValueType, IndexType >::mpi\_communicator, SchwarzWrappers::Metadata< ValueType, IndexType >::my\_local\_rank, SchwarzWrappers::Metadata< ValueType, IndexType >::my\_rank, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::neighbors\_in, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::neighbors\_out, SchwarzWrappers::Metadata< ValueType, IndexType >::num\_subdomains, SchwarzWrappers::Metadata< ValueType, IndexType >::permutation, and SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::put\_displacements.

```

50 : Initialize<ValueType, IndexType>(settings, metadata),
51   settings(settings),
52   metadata(metadata)
53 {
54     using vec_itype = gko::Array<IndexType>;
55     using vec_vecshared = gko::Array<IndexType *>;
56     metadata.my_local_rank =
57         Utils<ValueType, IndexType>::get_local_rank(metadata.mpi_communicator);
58     metadata.local_num_procs = Utils<ValueType, IndexType>::get_local_num_procs(
59         metadata.mpi_communicator);
60     auto my_local_rank = metadata.my_local_rank;
61     if (settings.executor_string == "omp") {
62         settings.executor = gko::OmpExecutor::create();
63         auto exec_info =
64             static_cast<gko::OmpExecutor *>(settings.executor.get())
65             ->get_exec_info();
66         exec_info->bind_to_core(metadata.my_local_rank);
67     } else if (settings.executor_string == "cuda") {
68         int num_devices = 0;
69         #if SCHW_HAVE_CUDA
70             SCHWARZ_ASSERT_NO_CUDA_ERRORS(cudaGetDeviceCount(&num_devices));
71         #else
72             SCHWARZ_NOT_IMPLEMENTED;
73         #endif
74         if (num_devices > 0) {
75             if (metadata.my_rank == 0) {
76                 std::cout << " Number of available devices: " << num_devices
77                     << std::endl;
78             }
79         } else {
80             std::cout << " No CUDA devices available for rank "
81                 << metadata.my_rank << std::endl;
82             std::exit(-1);
83         }
84     }
85     settings.executor = gko::CudaExecutor::create(
86         my_local_rank, gko::OmpExecutor::create());
87     auto exec_info = static_cast<gko::OmpExecutor *>(
88         settings.executor->get_master().get())
89         ->get_exec_info();
90     exec_info->bind_to_core(my_local_rank);
91     settings.cuda_device_guard =

```

```

92         std::make_shared<SchwarzWrappers::device_guard>(my_local_rank);
93
94         std::cout << " Rank " << metadata.my_rank << " with local rank "
95                 << my_local_rank << " has "
96                 << (static_cast<gko::CudaExecutor*>(settings.executor.get()))
97                     ->get_device_id()
98                 << " id of gpu" << std::endl;
99         MPI_Barrier(metadata.mpi_communicator);
100     } else if (settings.executor_string == "reference") {
101         settings.executor = gko::ReferenceExecutor::create();
102         auto exec_info =
103             static_cast<gko::ReferenceExecutor*>(settings.executor.get())
104             ->get_exec_info();
105         exec_info->bind_to_core(my_local_rank);
106     }
107
108     auto my_rank = this->metadata.my_rank;
109     auto comm_size = this->metadata.comm_size;
110     auto num_subdomains = this->metadata.num_subdomains;
111     auto global_size = this->metadata.global_size;
112
113     // Some arrays for partitioning and local matrix creation.
114     metadata.first_row = std::shared_ptr<vec_itype>(
115         new vec_itype(settings.executor->get_master(), num_subdomains + 1),
116         std::default_delete<vec_itype>());
117     metadata.permutation = std::shared_ptr<vec_itype>(
118         new vec_itype(settings.executor->get_master(), global_size),
119         std::default_delete<vec_itype>());
120     metadata.i_permutation = std::shared_ptr<vec_itype>(
121         new vec_itype(settings.executor->get_master(), global_size),
122         std::default_delete<vec_itype>());
123     metadata.global_to_local = std::shared_ptr<vec_itype>(
124         new vec_itype(settings.executor->get_master(), global_size),
125         std::default_delete<vec_itype>());
126     metadata.local_to_global = std::shared_ptr<vec_itype>(
127         new vec_itype(settings.executor->get_master(), global_size),
128         std::default_delete<vec_itype>());
129
130     // Some arrays for communication.
131     comm_struct.local_neighbors_in = std::shared_ptr<vec_itype>(
132         new vec_itype(settings.executor->get_master(), num_subdomains + 1),
133         std::default_delete<vec_itype>());
134     comm_struct.local_neighbors_out = std::shared_ptr<vec_itype>(
135         new vec_itype(settings.executor->get_master(), num_subdomains + 1),
136         std::default_delete<vec_itype>());
137     comm_struct.neighbors_in = std::shared_ptr<vec_itype>(
138         new vec_itype(settings.executor->get_master(), num_subdomains + 1),
139         std::default_delete<vec_itype>());
140     comm_struct.neighbors_out = std::shared_ptr<vec_itype>(
141         new vec_itype(settings.executor->get_master(), num_subdomains + 1),
142         std::default_delete<vec_itype>());
143     comm_struct.is_local_neighbor = std::vector<bool>(
144         num_subdomains + 1, 0);
145     comm_struct.global_get = std::shared_ptr<vec_vecshared>(
146         new vec_vecshared(settings.executor->get_master(), num_subdomains + 1),
147         std::default_delete<vec_vecshared>());
148     comm_struct.global_put = std::shared_ptr<vec_vecshared>(
149         new vec_vecshared(settings.executor->get_master(), num_subdomains + 1),
150         std::default_delete<vec_vecshared>());
151     // Need this to initialize the arrays with zeros.
152     std::vector<IndexType> temp(num_subdomains + 1, 0);
153     comm_struct.get_displacements = std::shared_ptr<vec_itype>(
154         new vec_itype(settings.executor->get_master(), temp.begin(),
155             temp.end()),
156         std::default_delete<vec_itype>());
157     comm_struct.put_displacements = std::shared_ptr<vec_itype>(
158         new vec_itype(settings.executor->get_master(), temp.begin(),
159             temp.end()),
160         std::default_delete<vec_itype>());

```

## 7.12.3 Member Function Documentation

### 7.12.3.1 print\_matrix()

```

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
void SchwarzWrappers::SchwarzBase<ValueType, IndexType>::print_matrix (

```



```
const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & matrix,
int rank,
std::string name )
```

The auxiliary function that prints a passed in CSR matrix.

#### Parameters

<i>matrix</i>	The matrix to be printed.
<i>subd</i>	The subdomain on which the vector exists.
<i>name</i>	The name of the matrix as a string.

#### 7.12.3.2 print\_vector()

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
void SchwarzWrappers::SchwarzBase< ValueType, IndexType >::print_vector (
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & vector,
    int subd,
    std::string name )
```

The auxiliary function that prints a passed in vector.

#### Parameters

<i>vector</i>	The vector to be printed.
<i>subd</i>	The subdomain on which the vector exists.
<i>name</i>	The name of the vector as a string.

#### 7.12.3.3 run()

```
template<typename ValueType , typename IndexType >
void SchwarzWrappers::SchwarzBase< ValueType, IndexType >::run (
    std::shared_ptr< gko::matrix::Dense< ValueType >> & solution )
```

The function that runs the actual solver and obtains the final solution.

#### Parameters

<i>solution</i>	The solution vector.
-----------------	----------------------

References SchwarzWrappers::Communicate< ValueType, IndexType >::exchange\_boundary(), SchwarzWrappers::Settings::executor, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::global\_matrix, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::global\_rhs, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::interface\_matrix, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\_inv\_perm, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\_matrix, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\_perm, SchwarzWrappers::SchwarzBase< ValueType, IndexType

>::local\_rhs, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\_solution, SchwarzWrappers::Communicate< ValueType, IndexType >::setup\_windows(), SchwarzWrappers::SchwarzBase< ValueType, IndexType >::triangular\_factor, and SchwarzWrappers::Communicate< ValueType, IndexType >::update\_boundary().

```

335 {
336     using vec_vtype = gko::matrix::Dense<ValueType>;
337     // The main solution vector
338     std::shared_ptr<vec_vtype> solution_vector = vec_vtype::create(
339         settings.executor, gko::dim<2>(metadata.global_size, 1));
340     // A temp local solution
341     std::shared_ptr<vec_vtype> init_guess =
342         vec_vtype::create(settings.executor, this->local_solution->get_size());
343     // A global gathered solution of the previous iteration.
344     std::shared_ptr<vec_vtype> global_old_solution = vec_vtype::create(
345         settings.executor, gko::dim<2>(metadata.global_size, 1));
346     // Setup the windows for the onesided communication.
347     this->setup_windows(settings, metadata, solution_vector);
348
349     const auto solver_settings =
350         (Settings::local_solver_settings::direct_solver_cholmod |
351          Settings::local_solver_settings::direct_solver_ginkgo |
352          Settings::local_solver_settings::iterative_solver_dealii |
353          Settings::local_solver_settings::iterative_solver_ginkgo) &
354         settings.local_solver;
355
356     ValueType local_residual_norm = -1.0, local_residual_norm0 = -1.0,
357         global_residual_norm = 0.0, global_residual_norm0 = -1.0;
358     metadata.iter_count = 0;
359     auto start_time = std::chrono::steady_clock::now();
360     int num_converged_procs = 0;
361
362     for (; metadata.iter_count < metadata.max_iters; ++(metadata.iter_count)) {
363         // Exchange the boundary values. The communication part.
364         MEASURE_ELAPSED_FUNC_TIME(
365             this->exchange_boundary(settings, metadata, solution_vector), 0,
366             metadata.my_rank, boundary_exchange, metadata.iter_count);
367
368         // Update the boundary and interior values after the exchanging from
369         // other processes.
370         MEASURE_ELAPSED_FUNC_TIME(
371             this->update_boundary(settings, metadata, this->
local_solution,
372                                     this->local_rhs, solution_vector,
373                                     global_old_solution, this->interface_matrix),
374             1, metadata.my_rank, boundary_update, metadata.iter_count);
375
376         // Check for the convergence of the solver.
377         num_converged_procs = 0;
378         MEASURE_ELAPSED_FUNC_TIME(
379             (Solve<ValueType, IndexType>::check_convergence(
380                 settings, metadata, this->comm_struct, this->convergence_vector,
381                 global_old_solution, this->local_solution, this->
local_matrix,
382                 local_residual_norm, local_residual_norm0, global_residual_norm,
383                 global_residual_norm0, num_converged_procs)),
384             2, metadata.my_rank, convergence_check, metadata.iter_count);
385
386         // break if the solution diverges.
387         if (std::isnan(global_residual_norm) || global_residual_norm > 1e12) {
388             std::cout << " Rank " << metadata.my_rank << " diverged in "
389                 << metadata.iter_count << " iters " << std::endl;
390             std::exit(-1);
391         }
392
393         // break if all processes detect that all other processes have
394         // converged otherwise continue iterations.
395         if (num_converged_procs == metadata.num_subdomains) {
396             break;
397         } else {
398             MEASURE_ELAPSED_FUNC_TIME(
399                 (Solve<ValueType, IndexType>::local_solve(
400                     settings, metadata, this->triangular_factor,
401                     this->local_perm, this->local_inv_perm, init_guess,
402                     this->local_solution)),
403                 3, metadata.my_rank, local_solve, metadata.iter_count);
404             // init_guess->copy_from(this->local_solution.get());
405             // Gather the local vector into the locally global vector for
406             // communication.
407             MEASURE_ELAPSED_FUNC_TIME(
408                 (Communicate<ValueType, IndexType>::local_to_global_vector
(
409                     settings, metadata, this->local_solution, solution_vector)),
410                 4, metadata.my_rank, expand_local_vec, metadata.iter_count);

```

```

411     }
412 }
413 MPI_Barrier(MPI_COMM_WORLD);
414 auto elapsed_time = std::chrono::duration<ValueType>(
415     std::chrono::steady_clock::now() - start_time);
416 std::cout << " Rank " << metadata.my_rank << " converged in "
417     << metadata.iter_count << " iters " << std::endl;
418 ValueType mat_norm = -1.0, rhs_norm = -1.0, sol_norm = -1.0,
419     residual_norm = -1.0;
420 // Compute the final residual norm. Also gathers the solution from all
421 // subdomains.
422 Solve<ValueType, IndexType>::compute_residual_norm(
423     settings, metadata, global_matrix, global_rhs, solution_vector,
424     mat_norm, rhs_norm, sol_norm, residual_norm);
425 gather_comm_data<ValueType, IndexType>(
426     metadata.num_subdomains, this->comm_struct, metadata.comm_data_struct);
427 // clang-format off
428 if (metadata.my_rank == 0)
429 {
430     std::cout
431         << " residual norm " << residual_norm << "\n"
432         << " relative residual norm of solution " << residual_norm/rhs_norm << "\n"
433         << " Time taken for solve " << elapsed_time.count()
434         << std::endl;
435     if (num_converged_procs < metadata.num_subdomains)
436     {
437         std::cout << " Did not converge in " << metadata.iter_count
438             << " iterations."
439             << std::endl;
440     }
441 }
442 // clang-format on
443 if (metadata.my_rank == 0) {
444     solution->copy_from(solution_vector.get());
445 }
446
447 // Communicate<ValueType, IndexType>::clear(settings);
448 }

```

The documentation for this class was generated from the following files:

- schwarz\_base.hpp (7ee6ebb)
- /home/runner/work/schwarz-lib/schwarz-lib/source/schwarz\_base.cpp (7ee6ebb)

## 7.13 SchwarzWrappers::Settings Struct Reference

The struct that contains the solver settings and the parameters to be set by the user.

```
#include <settings.hpp>
```

### Classes

- struct [comm\\_settings](#)  
*The settings for the various available communication paradigms.*
- struct [convergence\\_settings](#)  
*The various convergence settings available.*

### Public Types

- enum [partition\\_settings](#)  
*The partition algorithm to be used for partitioning the matrix.*
- enum [local\\_solver\\_settings](#)  
*The local solver algorithm for the local subdomain solves.*

## Public Attributes

- `std::string` `executor_string`  
*The string that contains the ginkgo executor paradigm.*
- `std::shared_ptr< gko::Executor >` `executor` = `gko::ReferenceExecutor::create()`  
*The ginkgo executor the code is to be executed on.*
- `std::shared_ptr< device_guard >` `cuda_device_guard`  
*The ginkgo executor the code is to be executed on.*
- `gko::int32` `overlap` = 2  
*The overlap between the subdomains.*
- `bool` `explicit_laplacian` = true  
*Flag if the laplacian matrix should be generated within the library.*
- `bool` `enable_random_rhs` = false  
*Flag to enable a random rhs.*
- `bool` `print_matrices` = false  
*Flag to enable printing of matrices.*
- `bool` `debug_print` = false  
*Flag to enable some debug printing.*
- `bool` `naturally_ordered_factor` = false  
*Disables the re-ordering of the matrix before computing the triangular factors during the CHOLMOD factorization.*
- `std::string` `metis_objtype`  
*This setting defines the objective type for the metis partitioning.*
- `bool` `use_precond` = false  
*Enable the block jacobi local preconditioner for the local solver.*
- `bool` `write_debug_out` = false  
*Enable the writing of debug out to file.*
- `bool` `write_perm_data` = false  
*Enable the local permutations from CHOLMOD to a file.*
- `int` `shifted_iter` = 1  
*Iteration shift for node local communication.*
- `std::string` `reorder`  
*The reordering for the local solve.*

### 7.13.1 Detailed Description

The struct that contains the solver settings and the parameters to be set by the user.

settings

### 7.13.2 Member Data Documentation

#### 7.13.2.1 explicit\_laplacian

```
bool SchwarzWrappers::Settings::explicit_laplacian = true
```

Flag if the laplacian matrix should be generated within the library.

If false, an external matrix and rhs needs to be provided

Referenced by `SchwarzWrappers::SchwarzBase< ValueType, IndexType >::initialize()`.

## 7.13.2.2 naturally\_ordered\_factor

```
bool SchwarzWrappers::Settings::naturally_ordered_factor = false
```

Disables the re-ordering of the matrix before computing the triangular factors during the CHOLMOD factorization.

**Note**

This is mainly to allow compatibility with GPU solution.

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

- settings.hpp (7ee6ebb)

## 7.14 SchwarzWrappers::Solve&lt; ValueType, IndexType &gt; Class Template Reference

The Solver class the provides the solver and the convergence checking methods.

```
#include <solve.hpp>
```

**Additional Inherited Members**

## 7.14.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class SchwarzWrappers::Solve< ValueType, IndexType >
```

The Solver class the provides the solver and the convergence checking methods.

**Template Parameters**

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

[Solve](#)

The documentation for this class was generated from the following files:

- solve.hpp (7ee6ebb)
- /home/runner/work/schwarz-lib/schwarz-lib/source/solve.cpp (7ee6ebb)

## 7.15 SchwarzWrappers::SolverRAS&lt; ValueType, IndexType &gt; Class Template Reference

An implementation of the solver interface using the RAS solver.

```
#include <restricted_schwarz.hpp>
```

## Public Member Functions

- [SolverRAS](#) ([Settings](#) &settings, [Metadata](#)< ValueType, IndexType > &metadata)  
*The constructor that takes in the user settings and a metadata struct containing the solver metadata.*
- void [setup\\_local\\_matrices](#) ([Settings](#) &settings, [Metadata](#)< ValueType, IndexType > &metadata, std::vector< unsigned int > &[partition\\_indices](#), std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &[global\\_matrix](#), std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &[local\\_matrix](#), std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &[interface\\_matrix](#), std::shared\_ptr< gko::matrix::Permutation< IndexType >> &[local\\_perm](#), std::shared\_ptr< gko::matrix::Permutation< IndexType >> &[local\\_inv\\_perm](#)) override  
*Sets up the local and the interface matrices from the global matrix and the partition indices.*
- void [setup\\_comm\\_buffers](#) () override  
*Sets up the communication buffers needed for the boundary exchange.*
- void [setup\\_windows](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &[main\\_buffer](#)) override  
*Sets up the windows needed for the asynchronous communication.*
- void [exchange\\_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &[solution\\_vector](#)) override  
*Exchanges the elements of the solution vector.*
- void [update\\_boundary](#) (const [Settings](#) &settings, const [Metadata](#)< ValueType, IndexType > &metadata, std::shared\_ptr< gko::matrix::Dense< ValueType >> &[local\\_solution](#), const std::shared\_ptr< gko::matrix::Dense< ValueType >> &[local\\_rhs](#), const std::shared\_ptr< gko::matrix::Dense< ValueType >> &[solution\\_vector](#), std::shared\_ptr< gko::matrix::Dense< ValueType >> &[global\\_old\\_solution](#), const std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> &[interface\\_matrix](#)) override  
*Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.*

## Additional Inherited Members

### 7.15.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class SchwarzWrappers::SolverRAS< ValueType, IndexType >
```

An implementation of the solver interface using the RAS solver.

#### Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 SolverRAS()

```
template<typename ValueType , typename IndexType >
SchwarzWrappers::SolverRAS< ValueType, IndexType >::SolverRAS (
```

```

    Settings & settings,
    Metadata< ValueType, IndexType > & metadata )

```

The constructor that takes in the user settings and a metadata struct containing the solver metadata.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>data</i>	The additional data struct.

```

50      : SchwarzBase<ValueType, IndexType>(settings, metadata)
51 {}

```

### 7.15.3 Member Function Documentation

#### 7.15.3.1 exchange\_boundary()

```

template<typename ValueType , typename IndexType >
void SchwarzWrappers::SolverRAS< ValueType, IndexType >::exchange_boundary (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & solution_vector ) [override],
[virtual]

```

Exchanges the elements of the solution vector.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>solution_vector</i>	The solution vector being exchanged between the subdomains.

Implements [SchwarzWrappers::Communicate< ValueType, IndexType >](#).

References [SchwarzWrappers::Settings::comm\\_settings::enable\\_onesided](#).

```

795 {
796     if (settings.comm_settings.enable_onesided) {
797         exchange_boundary_onesided<ValueType, IndexType>(
798             settings, metadata, this->comm_struct, solution_vector);
799     } else {
800         exchange_boundary_twosided<ValueType, IndexType>(
801             settings, metadata, this->comm_struct, solution_vector);
802     }
803 }

```

### 7.15.3.2 setup\_local\_matrices()

```
template<typename ValueType , typename IndexType >
void SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_local_matrices (
    Settings & settings,
    Metadata< ValueType, IndexType > & metadata,
    std::vector< unsigned int > & partition_indices,
    std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & global_matrix,
    std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & local_matrix,
    std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_matrix,
    std::shared_ptr< gko::matrix::Permutation< IndexType >> & local_perm,
    std::shared_ptr< gko::matrix::Permutation< IndexType >> & local_inv_perm ) [override],
[virtual]
```

Sets up the local and the interface matrices from the global matrix and the partition indices.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>partition_indices</i>	The array containing the partition indices.
<i>global_matrix</i>	The global system matrix.
<i>local_matrix</i>	The local system matrix.
<i>interface_matrix</i>	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.
<i>local_perm</i>	The local permutation, obtained through RCM or METIS.

Implements [SchwarzWrappers::Initialize< ValueType, IndexType >](#).

References [SchwarzWrappers::Metadata< ValueType, IndexType >::comm\\_size](#), [SchwarzWrappers::Settings](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::first\\_row](#), [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::global\\_matrix](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::global\\_size](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::global\\_to\\_local](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::i\\_permutation](#), [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::interface\\_matrix](#), [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\\_matrix](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::local\\_size](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::local\\_size\\_o](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::local\\_size\\_x](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::local\\_to\\_global](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::my\\_rank](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::num\\_subdomains](#), [SchwarzWrappers::Settings::overlap](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::overlap\\_row](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::overlap\\_size](#), and [SchwarzWrappers::Metadata< ValueType, IndexType >::permutation](#).

```
63 {
64     using mtx = gko::matrix::Csr<ValueType, IndexType>;
65     using vec_type = gko::Array<IndexType>;
66     using perm_type = gko::matrix::Permutation<IndexType>;
67     using arr = gko::Array<IndexType>;
68     auto my_rank = metadata.my_rank;
69     auto comm_size = metadata.comm_size;
70     auto num_subdomains = metadata.num_subdomains;
71     auto global_size = metadata.global_size;
72     auto mpi_itype = boost::mpi::get_mpi_datatype(*partition_indices.data());
73
74     MPI_Bcast(partition_indices.data(), global_size, mpi_itype, 0,
75             MPI_COMM_WORLD);
76
77     std::vector<IndexType> local_p_size(num_subdomains);
78     auto global_to_local = metadata.global_to_local->get_data();
79     auto local_to_global = metadata.local_to_global->get_data();
80
81     auto first_row = metadata.first_row->get_data();
```



```

82     auto permutation = metadata.permutation->get_data();
83     auto i_permutation = metadata.i_permutation->get_data();
84
85     auto nb = (global_size + num_subdomains - 1) /
num_subdomains;
86     auto partition_settings =
87     (Settings::partition_settings::partition_zoltan |
88     Settings::partition_settings::partitionmetis |
89     Settings::partition_settings::partition_regular |
90     Settings::partition_settings::partition_regular2d |
91     Settings::partition_settings::partition_custom) &
92     settings.partition;
93
94     IndexType *gmat_row_ptrs = global_matrix->get_row_ptrs();
95     IndexType *gmat_col_idxes = global_matrix->get_col_idxes();
96     ValueType *gmat_values = global_matrix->get_values();
97
98     // default local p size set for 1 subdomain.
99     first_row[0] = 0;
100     for (auto p = 0; p < num_subdomains; ++p) {
101         local_p_size[p] = std::min(global_size - first_row[p], nb);
102         first_row[p + 1] = first_row[p] + local_p_size[p];
103     }
104
105     if (partition_settings == Settings::partition_settings::partitionmetis ||
106         partition_settings ==
107         Settings::partition_settings::partition_regular2d) {
108         if (num_subdomains > 1) {
109             for (auto p = 0; p < num_subdomains; p++) {
110                 local_p_size[p] = 0;
111             }
112             for (auto i = 0; i < global_size; i++) {
113                 local_p_size[partition_indices[i]]++;
114             }
115             first_row[0] = 0;
116             for (auto p = 0; p < num_subdomains; ++p) {
117                 first_row[p + 1] = first_row[p] + local_p_size[p];
118             }
119             // permutation
120             for (auto i = 0; i < global_size; i++) {
121                 permutation[first_row[partition_indices[i]]] = i;
122                 first_row[partition_indices[i]]++;
123             }
124             for (auto p = num_subdomains; p > 0; p--) {
125                 first_row[p] = first_row[p - 1];
126             }
127             first_row[0] = 0;
128
129             // iperm
130             for (auto i = 0; i < global_size; i++) {
131                 i_permutation[permutation[i]] = i;
132             }
133         }
134
135         auto gmat_temp = mtx::create(settings.executor->get_master(),
136                                     global_matrix->get_size(),
137                                     global_matrix->get_num_stored_elements());
138
139         auto nnz = 0;
140         gmat_temp->get_row_ptrs()[0] = 0;
141         for (auto row = 0; row < metadata.global_size; ++row) {
142             for (auto col = gmat_row_ptrs[permutation[row]];
143                 col < gmat_row_ptrs[permutation[row] + 1]; ++col) {
144                 gmat_temp->get_col_idxes()[nnz] =
145                     i_permutation[gmat_col_idxes[col]];
146                 gmat_temp->get_values()[nnz] = gmat_values[col];
147                 nnz++;
148             }
149             gmat_temp->get_row_ptrs()[row + 1] = nnz;
150         }
151         global_matrix->copy_from(gmat_temp.get());
152
153         for (auto i = 0; i < global_size; i++) {
154             global_to_local[i] = 0;
155             local_to_global[i] = 0;
156         }
157         auto num = 0;
158         for (auto i = first_row[my_rank]; i < first_row[
159 my_rank + 1]; i++) {
160             global_to_local[i] = 1 + num;
161             local_to_global[num] = i;
162             num++;
163         }
164
165         IndexType old = 0;
166         for (auto k = 1; k < settings.overlap; k++) {
167             auto now = num;
168             for (auto i = old; i < now; i++) {

```

```

167         for (auto j = gmat_row_ptrs[local_to_global[i]];
168             j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
169             if (global_to_local[gmat_col_idxs[j]] == 0) {
170                 local_to_global[num] = gmat_col_idxs[j];
171                 global_to_local[gmat_col_idxs[j]] = 1 + num;
172                 num++;
173             }
174         }
175     }
176     old = now;
177 }
178 metadata.local_size = local_p_size[my_rank];
179 metadata.local_size_x = num;
180 metadata.local_size_o = global_size;
181 auto local_size = metadata.local_size;
182 auto local_size_x = metadata.local_size_x;
183
184 metadata.overlap_size = num - metadata.local_size;
185 metadata.overlap_row = std::shared_ptr<vec_itype>(
186     new vec_itype(gko::Array<IndexType>::view(
187         settings.executor, metadata.overlap_size,
188         &(metadata.local_to_global->get_data()[metadata.local_size])),
189     std::default_delete<vec_itype>());
190
191 auto nnz_local = 0;
192 auto nnz_interface = 0;
193
194 for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
195     for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; j++) {
196         if (global_to_local[gmat_col_idxs[j]] != 0) {
197             nnz_local++;
198         } else {
199             std::cout << " debug: invalid edge?" << std::endl;
200         }
201     }
202 }
203 auto temp = 0;
204 for (auto k = 0; k < metadata.overlap_size; k++) {
205     temp = metadata.overlap_row->get_data()[k];
206     for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1]; j++) {
207         if (global_to_local[gmat_col_idxs[j]] != 0) {
208             nnz_local++;
209         } else {
210             nnz_interface++;
211         }
212     }
213 }
214
215 std::shared_ptr<mtx> local_matrix_compute;
216 local_matrix_compute = mtx::create(settings.executor->get_master(),
217     gko::dim<2>(local_size_x), nnz_local);
218 IndexType *lmat_row_ptrs = local_matrix_compute->get_row_ptrs();
219 IndexType *lmat_col_idxs = local_matrix_compute->get_col_idxs();
220 ValueType *lmat_values = local_matrix_compute->get_values();
221
222 std::shared_ptr<mtx> interface_matrix_compute;
223 if (nnz_interface > 0) {
224     interface_matrix_compute =
225         mtx::create(settings.executor->get_master(),
226             gko::dim<2>(local_size_x), nnz_interface);
227 } else {
228     interface_matrix_compute = mtx::create(settings.executor->get_master());
229 }
230
231 IndexType *imat_row_ptrs = interface_matrix_compute->get_row_ptrs();
232 IndexType *imat_col_idxs = interface_matrix_compute->get_col_idxs();
233 ValueType *imat_values = interface_matrix_compute->get_values();
234
235 num = 0;
236 nnz_local = 0;
237 auto nnz_interface_temp = 0;
238 lmat_row_ptrs[0] = nnz_local;
239 if (nnz_interface > 0) {
240     imat_row_ptrs[0] = nnz_interface_temp;
241 }
242 // Local interior matrix
243 for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
244     for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; ++j) {
245         if (global_to_local[gmat_col_idxs[j]] != 0) {
246             lmat_col_idxs[nnz_local] =
247                 global_to_local[gmat_col_idxs[j]] - 1;
248             lmat_values[nnz_local] = gmat_values[j];
249             nnz_local++;
250         }
251     }
252     if (nnz_interface > 0) {
253         imat_row_ptrs[num + 1] = nnz_interface_temp;

```

```

254     }
255     lmat_row_ptrs[num + 1] = nnz_local;
256     num++;
257 }
258
259 // Interface matrix
260 if (nnz_interface > 0) {
261     nnz_interface = 0;
262     for (auto k = 0; k < metadata.overlap_size; k++) {
263         temp = metadata.overlap_row->get_data()[k];
264         for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1];
265             j++) {
266             if (global_to_local[gmat_col_idxs[j]] != 0) {
267                 lmat_col_idxs[nnz_local] =
268                     global_to_local[gmat_col_idxs[j]] - 1;
269                 lmat_values[nnz_local] = gmat_values[j];
270                 nnz_local++;
271             } else {
272                 imat_col_idxs[nnz_interface] = gmat_col_idxs[j];
273                 imat_values[nnz_interface] = gmat_values[j];
274                 nnz_interface++;
275             }
276         }
277         lmat_row_ptrs[num + 1] = nnz_local;
278         imat_row_ptrs[num + 1] = nnz_interface;
279         num++;
280     }
281 }
282 auto now = num;
283 for (auto i = old; i < now; i++) {
284     for (auto j = gmat_row_ptrs[local_to_global[i]];
285         j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
286         if (global_to_local[gmat_col_idxs[j]] == 0) {
287             local_to_global[num] = gmat_col_idxs[j];
288             global_to_local[gmat_col_idxs[j]] = 1 + num;
289             num++;
290         }
291     }
292 }
293
294 local_matrix = mtx::create(settings.executor);
295 local_matrix->copy_from(gko::lend(local_matrix_compute));
296 interface_matrix = mtx::create(settings.executor);
297 interface_matrix->copy_from(gko::lend(interface_matrix_compute));
298 }

```

### 7.15.3.3 setup\_windows()

```

template<typename ValueType , typename IndexType >
void SchwarzWrappers::SolverRAS< ValueType, IndexType >::setup_windows (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & main_buffer ) [override],
[virtual]

```

Sets up the windows needed for the asynchronous communication.

#### Parameters

<i>settings</i>	The settings struct.
<i>metadata</i>	The metadata struct.
<i>main_buffer</i>	The main buffer being exchanged between the subdomains.

Implements [SchwarzWrappers::Communicate< ValueType, IndexType >](#).

References [SchwarzWrappers::Settings::comm\\_settings::enable\\_get](#), [SchwarzWrappers::Settings::comm\\_↵  
settings::enable\\_lock\\_all](#), [SchwarzWrappers::Settings::comm\\_settings::enable\\_one\\_by\\_one](#), [SchwarzWrappers↵  
::Settings::comm\\_settings::enable\\_onesided](#), [SchwarzWrappers::Settings::comm\\_settings::enable\\_overlap](#),

SchwarzWrappers::Settings::comm\_settings::enable\_put, SchwarzWrappers::Settings::executor, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::get\_displacements, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::get\_request, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::global\_get, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::global\_put, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::is\_local\_neighbor, SchwarzWrappers::Metadata< ValueType, IndexType >::iter\_count, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_get, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_neighbors\_in, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_neighbors\_out, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_num\_neighbors\_in, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_num\_neighbors\_out, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::local\_put, SchwarzWrappers::Metadata< ValueType, IndexType >::local\_size\_o, SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\_solution, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::neighbors\_in, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::neighbors\_out, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::num\_neighbors\_in, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::num\_neighbors\_out, SchwarzWrappers::Metadata< ValueType, IndexType >::num\_subdomains, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::put\_displacements, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::put\_request, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::recv\_buffer, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::remote\_get, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::remote\_put, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::send\_buffer, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::window\_recv\_buffer, SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::window\_send\_buffer, and SchwarzWrappers::Communicate< ValueType, IndexType >::comm\_struct::window\_x.

```

502 {
503     using vec_itype = gko::Array<IndexType>;
504     using vec_vtype = gko::matrix::Dense<ValueType>;
505     auto num_subdomains = metadata.num_subdomains;
506     auto local_size_o = metadata.local_size_o;
507     auto neighbors_in = this->comm_struct.neighbors_in->get_data();
508     auto global_get = this->comm_struct.global_get->get_data();
509     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
510     auto global_put = this->comm_struct.global_put->get_data();
511
512     // set displacement for the MPI buffer
513     auto get_displacements = this->comm_struct.get_displacements->get_data();
514     auto put_displacements = this->comm_struct.put_displacements->get_data();
515     {
516         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
517         tmp_num_comm_elems[0] = 0;
518         for (auto j = 0; j < this->comm_struct.num_neighbors_in; j++) {
519             if ((global_get[j])[0] > 0) {
520                 int p = neighbors_in[j];
521                 tmp_num_comm_elems[p + 1] = (global_get[j])[0];
522             }
523         }
524         for (auto j = 0; j < num_subdomains; j++) {
525             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
526         }
527
528         auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
529         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, put_displacements,
530                     1, mpi_itype, MPI_COMM_WORLD);
531     }
532
533     {
534         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
535         tmp_num_comm_elems[0] = 0;
536         for (auto j = 0; j < this->comm_struct.num_neighbors_out; j++) {
537             if ((global_put[j])[0] > 0) {
538                 int p = neighbors_out[j];
539                 tmp_num_comm_elems[p + 1] = (global_put[j])[0];
540             }
541         }
542         for (auto j = 0; j < num_subdomains; j++) {
543             tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
544         }
545
546         auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
547         MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, get_displacements,
548                     1, mpi_itype, MPI_COMM_WORLD);
549     }
550 }

```

```

551 // setup windows
552 if (settings.comm_settings.enable_onesided) {
553     // Onesided
554     MPI_Win_create(main_buffer->get_values(),
555                   main_buffer->get_size()[0] * sizeof(ValueType),
556                   sizeof(ValueType), MPI_INFO_NULL, MPI_COMM_WORLD,
557                   &(this->comm_struct.window_x));
558 }
559
560
561 if (settings.comm_settings.enable_onesided) {
562     // MPI_Alloc_mem ? Custom allocator ? TODO
563     MPI_Win_create(this->local_residual_vector->get_values(),
564                   (num_subdomains) * sizeof(ValueType), sizeof(ValueType),
565                   MPI_INFO_NULL, MPI_COMM_WORLD,
566                   &(this->window_residual_vector));
567     std::vector<IndexType> zero_vec(num_subdomains, 0);
568     gko::Array<IndexType> temp_array(settings.executor->get_master(),
569                                     zero_vec.begin(), zero_vec.end());
570     this->convergence_vector = std::shared_ptr<vec_itype>(
571         new vec_itype(settings.executor->get_master(), temp_array),
572         std::default_delete<vec_itype>());
573     this->convergence_sent = std::shared_ptr<vec_itype>(
574         new vec_itype(settings.executor->get_master(), num_subdomains),
575         std::default_delete<vec_itype>());
576     this->convergence_local = std::shared_ptr<vec_itype>(
577         new vec_itype(settings.executor->get_master(), num_subdomains),
578         std::default_delete<vec_itype>());
579     MPI_Win_create(this->convergence_vector->get_data(),
580                   (num_subdomains) * sizeof(IndexType), sizeof(IndexType),
581                   MPI_INFO_NULL, MPI_COMM_WORLD,
582                   &(this->window_convergence));
583 }
584
585 if (settings.comm_settings.enable_onesided && num_subdomains > 1) {
586     // Lock all windows.
587     if (settings.comm_settings.enable_get &&
588         settings.comm_settings.enable_lock_all) {
589         MPI_Win_lock_all(0, this->comm_struct.window_send_buffer);
590     }
591     if (settings.comm_settings.enable_put &&
592         settings.comm_settings.enable_lock_all) {
593         MPI_Win_lock_all(0, this->comm_struct.window_recv_buffer);
594     }
595     if (settings.comm_settings.enable_one_by_one &&
596         settings.comm_settings.enable_lock_all) {
597         MPI_Win_lock_all(0, this->comm_struct.window_x);
598     }
599     MPI_Win_lock_all(0, this->window_residual_vector);
600     MPI_Win_lock_all(0, this->window_convergence);
601 }
602 }

```

#### 7.15.3.4 update\_boundary()

```

template<typename ValueType , typename IndexType >
void SchwarzWrappers::SolverRAS< ValueType, IndexType >::update_boundary (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & local_solution,
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & local_rhs,
    const std::shared_ptr< gko::matrix::Dense< ValueType >> & solution_vector,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_old_solution,
    const std::shared_ptr< gko::matrix::Csr< ValueType, IndexType >> & interface_←
matrix ) [override], [virtual]

```

Update the values into local vector from obtained from the neighboring sub-domains using the interface matrix.

#### Parameters

<i>settings</i>	The settings struct.
-----------------	----------------------

## Parameters

<i>metadata</i>	The metadata struct.
<i>local_solution</i>	The local solution vector in the subdomain.
<i>local_rhs</i>	The local right hand side vector in the subdomain.
<i>solution_vector</i>	The workspace solution vector.
<i>global_old_solution</i>	The global solution vector of the previous iteration.
<i>interface_matrix</i>	The interface matrix containing the interface and the overlap data mainly used for exchanging values between different sub-domains.

Implements [SchwarzWrappers::Communicate< ValueType, IndexType >](#).

References [SchwarzWrappers::Settings::executor](#), [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::interface\\_matrix](#), [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\\_rhs](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::local\\_size\\_x](#), [SchwarzWrappers::SchwarzBase< ValueType, IndexType >::local\\_solution](#), [SchwarzWrappers::Metadata< ValueType, IndexType >::num\\_subdomains](#), and [SchwarzWrappers::Settings::overlap](#).

```

815 {
816     using vec_vtype = gko::matrix::Dense<ValueType>;
817     auto one = gko::initialize<gko::matrix::Dense<ValueType>>(
818         {1.0}, settings.executor);
819     auto neg_one = gko::initialize<gko::matrix::Dense<ValueType>>(
820         {-1.0}, settings.executor);
821     auto local_size_x = metadata.local_size_x;
822     local_solution->copy_from(local_rhs.get());
823     global_old_solution->copy_from(solution_vector.get());
824     if (metadata.num_subdomains > 1 && settings.overlap > 0) {
825         auto temp_solution = vec_vtype::create(
826             settings.executor, local_solution->get_size(),
827             gko::Array<ValueType>::view(
828                 settings.executor, local_solution->get_size()[0],
829                 &(global_old_solution->get_values()[0])),
830             1);
831         interface_matrix->apply(neg_one.get(), temp_solution.get(), one.get(),
832                                (local_solution).get());
833     }
834 }
```

The documentation for this class was generated from the following files:

- [restricted\\_schwarz.hpp \(7ee6ebb\)](#)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/restricted\\_schwarz.cpp \(7ee6ebb\)](#)

## 7.16 SchwarzWrappers::Utils< ValueType, IndexType > Struct Template Reference

The utilities class which provides some checks and basic utilities.

```
#include <utils.hpp>
```

### 7.16.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
struct SchwarzWrappers::Utils< ValueType, IndexType >
```

The utilities class which provides some checks and basic utilities.

## Template Parameters

<i>ValueType</i>	The type of the floating point values.
<i>IndexType</i>	The type of the index type values.

[Utils](#)

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

- [utils.hpp \(7ee6ebb\)](#)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/utils.cpp \(7ee6ebb\)](#)





# Index

- BadDimension, [19](#)
  - BadDimension, [19](#)
- Communicate, [9](#)
- CudaError, [28](#)
  - CudaError, [29](#)
- CusparsError, [29](#)
  - CusparsError, [30](#)
- exchange\_boundary
  - SchwarzWrappers::Communicate, [26](#)
  - SchwarzWrappers::SolverRAS, [49](#)
- explicit\_laplacian
  - SchwarzWrappers::Settings, [46](#)
- generate\_rhs
  - SchwarzWrappers::Initialize, [32](#)
- global\_get
  - SchwarzWrappers::Communicate::comm\_struct, [22](#)
- global\_put
  - SchwarzWrappers::Communicate::comm\_struct, [22](#)
- Initialization, [10](#)
- is\_local\_neighbor
  - SchwarzWrappers::Communicate::comm\_struct, [23](#)
- local\_get
  - SchwarzWrappers::Communicate::comm\_struct, [23](#)
- local\_put
  - SchwarzWrappers::Communicate::comm\_struct, [23](#)
- local\_solver\_tolerance
  - SchwarzWrappers::Metadata, [38](#)
- local\_to\_global\_vector
  - SchwarzWrappers::Communicate, [26](#)
- MetisError, [38](#)
  - MetisError, [38](#)
- naturally\_ordered\_factor
  - SchwarzWrappers::Settings, [46](#)
- partition
  - SchwarzWrappers::Initialize, [32](#)
- print\_matrix
  - SchwarzWrappers::SchwarzBase, [42](#)
- print\_vector
  - SchwarzWrappers::SchwarzBase, [43](#)
- ProcessTopology, [15](#)
- remote\_get
  - SchwarzWrappers::Communicate::comm\_struct, [24](#)
- remote\_put
  - SchwarzWrappers::Communicate::comm\_struct, [24](#)
- run
  - SchwarzWrappers::SchwarzBase, [43](#)
- Schwarz Class, [11](#)
- SchwarzBase
  - SchwarzWrappers::SchwarzBase, [40](#)
- SchwarzWrappers, [15](#)
  - SchwarzWrappers::CommHelpers, [16](#)
  - SchwarzWrappers::Communicate
    - exchange\_boundary, [26](#)
    - local\_to\_global\_vector, [26](#)
    - setup\_windows, [27](#)
    - update\_boundary, [27](#)
  - SchwarzWrappers::Communicate< ValueType, Index↔Type >, [25](#)
  - SchwarzWrappers::Communicate< ValueType, Index↔Type >::comm\_struct, [21](#)
  - SchwarzWrappers::Communicate::comm\_struct
    - global\_get, [22](#)
    - global\_put, [22](#)
    - is\_local\_neighbor, [23](#)
    - local\_get, [23](#)
    - local\_put, [23](#)
    - remote\_get, [24](#)
    - remote\_put, [24](#)
  - SchwarzWrappers::ConvergenceTools, [16](#)
  - SchwarzWrappers::Initialize
    - generate\_rhs, [32](#)
    - partition, [32](#)
    - setup\_global\_matrix\_laplacian, [33](#)
    - setup\_local\_matrices, [34](#)
    - setup\_vectors, [35](#)
  - SchwarzWrappers::Initialize< ValueType, IndexType >, [31](#)
  - SchwarzWrappers::Metadata
    - local\_solver\_tolerance, [38](#)
    - tolerance, [38](#)
  - SchwarzWrappers::Metadata< ValueType, IndexType >, [36](#)
  - SchwarzWrappers::PartitionTools, [17](#)
  - SchwarzWrappers::SchwarzBase

- print\_matrix, [42](#)
- print\_vector, [43](#)
- run, [43](#)
- SchwarzBase, [40](#)
- SchwarzWrappers::SchwarzBase< ValueType, IndexType >, [39](#)
- SchwarzWrappers::Settings, [45](#)
  - explicit\_laplacian, [46](#)
  - naturally\_ordered\_factor, [46](#)
- SchwarzWrappers::Settings::comm\_settings, [20](#)
- SchwarzWrappers::Settings::convergence\_settings, [28](#)
- SchwarzWrappers::Solve< ValueType, IndexType >, [47](#)
- SchwarzWrappers::SolverRAS< ValueType, IndexType >, [47](#)
- SchwarzWrappers::SolverRAS
  - exchange\_boundary, [49](#)
  - setup\_local\_matrices, [49](#)
  - setup\_windows, [53](#)
  - SolverRAS, [48](#)
  - update\_boundary, [55](#)
- SchwarzWrappers::SolverTools, [17](#)
- SchwarzWrappers::Utils< ValueType, IndexType >, [56](#)
- SchwarzWrappers::device\_guard, [30](#)
- setup\_global\_matrix\_laplacian
  - SchwarzWrappers::Initialize, [33](#)
- setup\_local\_matrices
  - SchwarzWrappers::Initialize, [34](#)
  - SchwarzWrappers::SolverRAS, [49](#)
- setup\_vectors
  - SchwarzWrappers::Initialize, [35](#)
- setup\_windows
  - SchwarzWrappers::Communicate, [27](#)
  - SchwarzWrappers::SolverRAS, [53](#)
- Solve, [12](#)
- SolverRAS
  - SchwarzWrappers::SolverRAS, [48](#)
- tolerance
  - SchwarzWrappers::Metadata, [38](#)
- update\_boundary
  - SchwarzWrappers::Communicate, [27](#)
  - SchwarzWrappers::SolverRAS, [55](#)
- Utils, [13](#)