

schwz

Generated automatically from fix-gpu-transfers

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	schwz Namespace Reference . . . . .	15
6.2.1	Detailed Description . . . . .	16
6.3	schwz::CommHelpers Namespace Reference . . . . .	16
6.3.1	Detailed Description . . . . .	16
6.4	schwz::conv_tools Namespace Reference . . . . .	16
6.4.1	Detailed Description . . . . .	16
6.5	schwz::PartitionTools Namespace Reference . . . . .	17
6.5.1	Detailed Description . . . . .	17
6.6	schwz::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	schwz::Settings::comm_settings Struct Reference . . . . .	20
7.2.1	Detailed Description . . . . .	21
7.3	schwz::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 . . . . .	23
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	schwz::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	schwz::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	schwz::device_guard Class Reference	30
7.8.1	Detailed Description	30
7.9	schwz::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()	33
7.9.2.4	setup_local_matrices()	35
7.9.2.5	setup_vectors()	35
7.10	schwz::Metadata< ValueType, IndexType > Struct Template Reference	36
7.10.1	Detailed Description	38
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	39
7.11.2	Constructor & Destructor Documentation	39
7.11.2.1	MetisError()	39
7.12	schwz::Metadata< ValueType, IndexType >::post_process_data Struct Reference	39
7.12.1	Detailed Description	40
7.13	schwz::SchwarzBase< ValueType, IndexType > Class Template Reference	40
7.13.1	Detailed Description	41
7.13.2	Constructor & Destructor Documentation	41
7.13.2.1	SchwarzBase()	41
7.13.3	Member Function Documentation	42

7.13.3.1	<a href="#">print_matrix()</a>	42
7.13.3.2	<a href="#">print_vector()</a>	44
7.13.3.3	<a href="#">run()</a>	44
7.14	<a href="#">schwz::Settings Struct Reference</a>	46
7.14.1	<a href="#">Detailed Description</a>	48
7.14.2	<a href="#">Member Data Documentation</a>	48
7.14.2.1	<a href="#">explicit_laplacian</a>	48
7.14.2.2	<a href="#">naturally_ordered_factor</a>	48
7.15	<a href="#">schwz::Solve&lt; ValueType, IndexType &gt; Class Template Reference</a>	48
7.15.1	<a href="#">Detailed Description</a>	48
7.16	<a href="#">schwz::SolverRAS&lt; ValueType, IndexType &gt; Class Template Reference</a>	49
7.16.1	<a href="#">Detailed Description</a>	49
7.16.2	<a href="#">Constructor &amp; Destructor Documentation</a>	50
7.16.2.1	<a href="#">SolverRAS()</a>	50
7.16.3	<a href="#">Member Function Documentation</a>	50
7.16.3.1	<a href="#">exchange_boundary()</a>	50
7.16.3.2	<a href="#">setup_local_matrices()</a>	51
7.16.3.3	<a href="#">setup_windows()</a>	54
7.16.3.4	<a href="#">update_boundary()</a>	57
7.17	<a href="#">UmfpackError Class Reference</a>	58
7.17.1	<a href="#">Detailed Description</a>	58
7.17.2	<a href="#">Constructor &amp; Destructor Documentation</a>	58
7.17.2.1	<a href="#">UmfpackError()</a>	58
7.18	<a href="#">schwz::Utils&lt; ValueType, IndexType &gt; Struct Template Reference</a>	59
7.18.1	<a href="#">Detailed Description</a>	59
<b>Index</b>		<b>61</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**



## 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

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

#### Classes

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

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

### Classes

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

- [schwz::conv\\_tools](#)  
*The [conv\\_tools](#) namespace .*
- [schwz::SolverTools](#)  
*The [SolverTools](#) namespace .*

### Classes

- struct [schwz::Metadata](#)< [ValueType](#), [IndexType](#) >  
*The solver metadata struct.*
- class [schwz::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 `schwz::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 schwz Namespace Reference

The Schwarz wrappers namespace.

#### Namespaces

- [CommHelpers](#)  
*The CommHelper namespace .*
- [conv\\_tools](#)  
*The conv\_tools 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 schwz::CommHelpers Namespace Reference

The CommHelper namespace .

### 6.3.1 Detailed Description

The CommHelper namespace .

comm\_helpers

## 6.4 schwz::conv\_tools Namespace Reference

The [conv\\_tools](#) namespace .

### 6.4.1 Detailed Description

The [conv\\_tools](#) namespace .

[conv\\_tools](#)

## 6.5 schwz::PartitionTools Namespace Reference

The [PartitionTools](#) namespace .

### 6.5.1 Detailed Description

The [PartitionTools](#) namespace .

part\_tools

## 6.6 schwz::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 (394983b)

## 7.2 schwz::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 (394983b)

## 7.3 schwz::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::matrix::Dense< ValueType > >` [cpu\\_send\\_buffer](#)  
*The send buffer used for the actual communication for both one-sided and two-sided.*
- `std::shared_ptr< gko::matrix::Dense< ValueType > >` [cpu\\_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\\_cpu\\_recv\\_buffer](#)  
*The RDMA window for the recv buffer.*
- `MPI_Win` [window\\_cpu\\_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 schwz::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 *> > schwz::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 `schwz::SchwarzBase< ValueType, IndexType >::initialize()`, `schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `schwz::SolverRAS< ValueType, IndexType >::setup_windows()`.

## 7.3.2.2 global\_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::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 `schwz::SchwarzBase< ValueType, IndexType >::initialize()`, `schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `schwz::SolverRAS< ValueType, IndexType >::setup_windows()`.

## 7.3.2.3 is\_local\_neighbor

```
template<typename ValueType , typename IndexType >
std::vector<bool> schwz::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 `schwz::SchwarzBase< ValueType, IndexType >::initialize()`, `schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `schwz::SolverRAS< ValueType, IndexType >::setup_windows()`.

## 7.3.2.4 local\_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::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 `schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `schwz::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.5 local\_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::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 `schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `schwz::SolverRAS< ValueType, IndexType >::setup_windows()`.

### 7.3.2.6 remote\_get

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::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←_put[p][i]` contains the index of the solution vector to be received from subdomain p.

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

### 7.3.2.7 remote\_put

```
template<typename ValueType , typename IndexType >
std::shared_ptr<gko::Array<IndexType *> > schwz::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 `schwz::SolverRAS< ValueType, IndexType >::setup_comm_buffers()`, and `schwz::SolverRAS< ValueType, IndexType >::setup_windows()`.

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

- `communicate.hpp` (394983b)

## 7.4 schwz::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 >> &global\_solution)=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 >> &global\_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 schwz::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 schwz::Communicate< ValueType, IndexType >::exchange_boundary (
    const Settings & settings,
    const Metadata< ValueType, IndexType > & metadata,
    std::shared_ptr< gko::matrix::Dense< ValueType >> & global_solution ) [pure
virtual]
```

Exchanges the elements of the solution vector.

#### Parameters

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

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

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

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

```
template<typename ValueType , typename IndexType >
void schwz::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.

```
69 {
70     using vec = gko::matrix::Dense<ValueType>;
71     auto alpha = gko::initialize<gko::matrix::Dense<ValueType>>(
72         {1.0}, settings.executor);
73     auto temp_vector = vec::create(
74         settings.executor, gko::dim<2>(metadata.local_size, 1),
```



```

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

```

#### 7.4.2.3 setup\_windows()

```

template<typename ValueType , typename IndexType >
void schwz::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 [schwz::SolverRAS< ValueType, IndexType >](#).

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

#### 7.4.2.4 update\_boundary()

```

template<typename ValueType , typename IndexType >
void schwz::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 >> & global_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>global_solution</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 [schwz::SolverRAS< ValueType, IndexType >](#).

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

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

- [communicate.hpp \(394983b\)](#)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/communicate.cpp \(394983b\)](#)

## 7.5 schwz::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 \(394983b\)](#)

## 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 (394983b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (394983b)

## 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 (394983b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (394983b)

## 7.8 schwz::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 (394983b)

## 7.9 schwz::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](#) (const std::string &filename, 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)  
*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)=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 schwz::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 schwz::Initialize< ValueType, IndexType >::generate_rhs (
    std::vector< ValueType > & rhs )
```

Generates the right hand side vector.

##### Parameters

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

References schwz::Initialize< ValueType, IndexType >::setup\_global\_matrix().

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

```
90 {
91     std::uniform_real_distribution<double> unif(0.0, 1.0);
92     std::default_random_engine engine;
93     for (gko::size_type i = 0; i < rhs.size(); ++i) {
94         rhs[i] = unif(engine);
95     }
96 }
```

#### 7.9.2.2 partition()

```
template<typename ValueType , typename IndexType >
void schwz::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 schwz::Metadata< ValueType, IndexType >::global\_size, schwz::Metadata< ValueType, IndexType >::my\_rank, schwz::Metadata< ValueType, IndexType >::num\_subdomains, and schwz::Settings::write\_debug\_out.

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

```

284 {
285     partition_indices.resize(metadata.global_size);
286     if (metadata.my_rank == 0) {
287         auto partition_settings =
288             (Settings::partition_settings::partition_zoltan |
289              Settings::partition_settings::partition_metis |
290              Settings::partition_settings::partition_regular |
291              Settings::partition_settings::partition_regular2d |
292              Settings::partition_settings::partition_custom) &
293             settings.partition;
294
295         if (partition_settings ==
296             Settings::partition_settings::partition_zoltan) {
297             SCHWARZ_NOT_IMPLEMENTED;
298         } else if (partition_settings ==
299             Settings::partition_settings::partition_metis) {
300             if (metadata.my_rank == 0) {
301                 std::cout << " METIS partition" << std::endl;
302             }
303             PartitionTools::PartitionMetis(
304                 settings, global_matrix, this->cell_weights,
305                 metadata.num_subdomains, partition_indices);
306         } else if (partition_settings ==
307             Settings::partition_settings::partition_regular) {
308             if (metadata.my_rank == 0) {
309                 std::cout << " Regular 1D partition" << std::endl;
310             }
311             PartitionTools::PartitionRegular(
312                 global_matrix, metadata.num_subdomains, partition_indices);
313         } else if (partition_settings ==
314             Settings::partition_settings::partition_regular2d) {
315             if (metadata.my_rank == 0) {
316                 std::cout << " Regular 2D partition" << std::endl;
317             }
318             PartitionTools::PartitionRegular2D(
319                 global_matrix, settings.write_debug_out,
320                 metadata.num_subdomains, partition_indices);
321         } else if (partition_settings ==
322             Settings::partition_settings::partition_custom) {
323             // User partitions mesh manually
324             SCHWARZ_NOT_IMPLEMENTED;
325         } else {
326             SCHWARZ_NOT_IMPLEMENTED;
327         }
328     }
329 }

```

### 7.9.2.3 setup\_global\_matrix()

```

template<typename ValueType , typename IndexType >
void schwz::Initialize< ValueType, IndexType >::setup_global_matrix (
    const std::string & filename,
    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 `schwz::Initialize< ValueType, IndexType >::generate_rhs()`, and `schwz::SchwarzBase< ValueType, IndexType >::initialize()`.

```

200 {
201     using index_type = IndexType;
202     using value_type = ValueType;
203     using mtx = gko::matrix::Csr<value_type, index_type>;
204     if (settings.matrix_filename != "null") {
205         auto input_file = std::ifstream(filename, std::ios::in);
206         if (!input_file) {
207             std::cerr << "Could not find the file \"" << filename
208                 << "\", which is required for this test.\n";
209         }
210         global_matrix =
211             gko::read<mtx>(input_file, settings.executor->get_master());
212         global_matrix->sort_by_column_index();
213         std::cout << "Matrix from file " << filename << std::endl;
214     } else if (settings.matrix_filename == "null" &&
215         settings.explicit_laplacian) {
216         std::cout << "Laplacian 2D Matrix (generated in house) " << std::endl;
217         gko::size_type global_size = oned_laplacian_size *
218             oned_laplacian_size;
219         global_matrix = mtx::create(settings.executor->get_master(),
220             gko::dim<2>(global_size), 5 * global_size);
221         value_type *values = global_matrix->get_values();
222         index_type *row_ptrs = global_matrix->get_row_ptrs();
223         index_type *col_idxs = global_matrix->get_col_idxs();
224
225         std::vector<gko::size_type> exclusion_set;
226
227         std::map<IndexType, ValueType> stencil_map = {
228             {-oned_laplacian_size, -1}, {-1, -1}, {0, 4}, {1, -1},
229             {oned_laplacian_size, -1},
230         };
231         for (auto i = 2; i < global_size; ++i) {
232             gko::size_type index = (i - 1) * oned_laplacian_size;
233             if (index * index < global_size * global_size) {
234                 exclusion_set.push_back(
235                     linearize_index(index, index - 1, global_size));
236                 exclusion_set.push_back(
237                     linearize_index(index - 1, index, global_size));
238             }
239         }
240
241         std::sort(exclusion_set.begin(),
242             exclusion_set.begin() + exclusion_set.size());
243
244         IndexType pos = 0;
245         IndexType col_idx = 0;
246         row_ptrs[0] = pos;
247         gko::size_type cur_idx = 0;
248         for (IndexType i = 0; i < global_size; ++i) {
249             for (auto ofs : stencil_map) {
250                 auto in_exclusion_flag =
251                     (exclusion_set[cur_idx] ==
252                     linearize_index(i, i + ofs.first, global_size));
253                 if (0 <= i + ofs.first && i + ofs.first < global_size &&
254                     !in_exclusion_flag) {
255                     values[pos] = ofs.second;
256                     col_idxs[pos] = i + ofs.first;
257                     ++pos;
258                 }
259                 if (in_exclusion_flag) {
260                     cur_idx++;
261                 }
262                 col_idx = row_ptrs[i + 1] - pos;
263             }
264             row_ptrs[i + 1] = pos;
265         }
266     } else {
267         std::cerr << " Need to provide a matrix or enable the default "
268             "laplacian matrix."
269             << std::endl;
270         std::exit(-1);
271     }
272 }

```



## 7.9.2.4 setup\_local\_matrices()

```

template<typename ValueType , typename IndexType >
void schwz::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 )
[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 [schwz::SolverRAS< ValueType, IndexType >](#).

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

## 7.9.2.5 setup\_vectors()

```

template<typename ValueType , typename IndexType >
void schwz::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 )

```

Setup the vectors with default values and allocate memory 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.

References `schwz::Settings::executor`, `schwz::Metadata< ValueType, IndexType >::first_row`, `schwz::Metadata< ValueType, IndexType >::local_size_x`, and `schwz::Metadata< ValueType, IndexType >::my_rank`.

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

```

339 {
340     using vec = gko::matrix::Dense<ValueType>;
341     auto my_rank = metadata.my_rank;
342     auto first_row = metadata.first_row->get_data()[my_rank];
343
344     // Copy the global rhs vector to the required executor.
345     gko::Array<ValueType> temp_rhs{settings.executor->get_master(), rhs.begin(),
346                                   rhs.end()};
347     global_rhs = vec::create(settings.executor,
348                             gko::dim<2>{metadata.global_size, 1}, temp_rhs, 1);
349
350     local_rhs =
351         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
352     // Extract the local rhs from the global rhs. Also takes into account the
353     // overlap.
354     SolverTools::extract_local_vector(settings, metadata, local_rhs.get(),
355                                       global_rhs.get(), first_row);
356
357     local_solution =
358         vec::create(settings.executor, gko::dim<2>(metadata.local_size_x, 1));
359 }
```

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

- `initialization.hpp` (394983b)
- `/home/runner/work/schwarz-lib/schwarz-lib/source/initialization.cpp` (394983b)

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

The solver metadata struct.

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

### Classes

- struct `post_process_data`  
*The struct used for storing data for post-processing.*

### 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 Schwarz solver.*

  - IndexType `local_max_iters`
- The maximum iteration count of the local iterative solver.*

  - std::string `local_precond`
- Local preconditioner.*

  - 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 schwz::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 schwz::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 schwz::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 (394983b)

## 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 (394983b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (394983b)

## 7.12 schwz::Metadata< ValueType, IndexType >::post\_process\_data Struct Reference

The struct used for storing data for post-processing.

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

### 7.12.1 Detailed Description

```
template<typename ValueType, typename IndexType>
struct schwz::Metadata< ValueType, IndexType >::post_process_data
```

The struct used for storing data for post-processing.

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

- settings.hpp (394983b)

## 7.13 schwz::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\\_l](#)  
*The local lower triangular factor used for the triangular solves.*
- std::shared\_ptr< gko::matrix::Csr< ValueType, IndexType >> [triangular\\_factor\\_u](#)  
*The local upper 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 > >` [work\\_vector](#)
- A work vector on the device.*
- `std::shared_ptr< gko::matrix::Dense< ValueType > >` [cpu\\_work\\_vector](#)
- The work vector on the CPU.*
- `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.*
- `std::vector< ValueType >` [local\\_residual\\_vector\\_out](#)
- The global residual vector.*
- `std::vector< std::vector< ValueType > >` [global\\_residual\\_vector\\_out](#)
- The local residual vector.*

## Additional Inherited Members

### 7.13.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class schwz::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.

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 SchwarzBase()

```
template<typename ValueType , typename IndexType >
schwz::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 `schwz::Settings::cuda_device_guard`, `schwz::Settings::executor`, `schwz::Settings::executor_string`, `schwz::Metadata< ValueType, IndexType >::local_num_procs`, `schwz::Metadata< ValueType, IndexType >::mpi_communicator`, `schwz::Metadata< ValueType, IndexType >::my_local_rank`, and `schwz::Metadata< ValueType, IndexType >::my_rank`.

```

74     : Initialize<ValueType, IndexType>(settings, metadata),
75     settings(settings),
76     metadata(metadata)
77 {
78     using vec_itype = gko::Array<IndexType>;
79     using vec_vecshared = gko::Array<IndexType *>;
80     metadata.my_local_rank =
81         Utils<ValueType, IndexType>::get_local_rank(metadata.mpi_communicator);
82     metadata.local_num_procs = Utils<ValueType, IndexType>::get_local_num_procs(
83         metadata.mpi_communicator);
84     auto my_local_rank = metadata.my_local_rank;
85     if (settings.executor_string == "omp") {
86         settings.executor = gko::OmpExecutor::create();
87         auto exec_info =
88             static_cast<gko::OmpExecutor *>(settings.executor.get())
89             ->get_exec_info();
90         exec_info->bind_to_core(metadata.my_local_rank);
91     } else if (settings.executor_string == "cuda") {
92         int num_devices = 0;
93         #if SCHW_HAVE_CUDA
94         SCHWARZ_ASSERT_NO_CUDA_ERRORS(cudaGetDeviceCount(&num_devices));
95         #else
96         SCHWARZ_NOT_IMPLEMENTED;
97         #endif
98         Utils<ValueType, IndexType>::assert_correct_cuda_devices(
99             num_devices, metadata.my_rank);
100         settings.executor = gko::CudaExecutor::create(
101             my_local_rank, gko::OmpExecutor::create());
102         auto exec_info = static_cast<gko::OmpExecutor *>(
103             settings.executor->get_master().get())
104             ->get_exec_info();
105         exec_info->bind_to_core(my_local_rank);
106         settings.cuda_device_guard =
107             std::make_shared<schwz::device_guard>(my_local_rank);
108         std::cout << " Rank " << metadata.my_rank << " with local rank "
109             << my_local_rank << " has "
110             << (static_cast<gko::CudaExecutor *>(settings.executor.get()))
111             ->get_device_id()
112             << " id of gpu" << std::endl;
113         MPI_Barrier(metadata.mpi_communicator);
114     } else if (settings.executor_string == "reference") {
115         settings.executor = gko::ReferenceExecutor::create();
116         auto exec_info =
117             static_cast<gko::ReferenceExecutor *>(settings.executor.get())
118             ->get_exec_info();
119         exec_info->bind_to_core(my_local_rank);
120     }
121 }
122 }
123 }

```

### 7.13.3 Member Function Documentation

#### 7.13.3.1 print\_matrix()

```

template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
void schwz::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.

7.13.3.2 `print_vector()`

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
void schwz::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.

7.13.3.3 `run()`

```
template<typename ValueType , typename IndexType >
void schwz::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 `schwz::SchwarzBase< ValueType, IndexType >::cpu_work_vector`, `schwz::Communicate< ValueType, IndexType >::exchange_boundary()`, `schwz::Settings::executor`, `schwz::SchwarzBase< ValueType, IndexType >::global_matrix`, `schwz::SchwarzBase< ValueType, IndexType >::global_rhs`, `schwz::SchwarzBase< ValueType, IndexType >::global_solution`, `schwz::SchwarzBase< ValueType, IndexType >::interface_matrix`, `schwz::SchwarzBase< ValueType, IndexType >::local_inv_perm`, `schwz::SchwarzBase< ValueType, IndexType >::local_matrix`, `schwz::SchwarzBase< ValueType, IndexType >::local_perm`, `schwz::SchwarzBase< ValueType, IndexType >::local_rhs`, `schwz::SchwarzBase< ValueType, IndexType >::local_solution`, `schwz::Communicate< ValueType, IndexType >::setup_windows()`, `schwz::SchwarzBase< ValueType, IndexType >::triangular_factor_l`, `schwz::SchwarzBase< ValueType, IndexType >::triangular_factor_u`, `schwz::Communicate< ValueType, IndexType >::update_boundary()`, `schwz::SchwarzBase< ValueType, IndexType >::work_vector`, and `schwz::Settings::write_iters_and_residuals`.

```

316 {
317     using vec_vtype = gko::matrix::Dense<ValueType>;
318     if (!solution.get()) {
319         solution =
320             vec_vtype::create(settings.executor->get_master(),
321                             gko::dim<2>(this->metadata.global_size, 1));
322     }
323     // The main solution vector
324     std::shared_ptr<vec_vtype> global_solution = vec_vtype::create(
325         this->settings.executor, gko::dim<2>(this->metadata.global_size, 1));
326     // Work vectors.
327     this->work_vector = vec_vtype::create(
328         settings.executor, gko::dim<2>(2 * this->metadata.local_size_x, 1));
329     this->cpu_work_vector =
330         vec_vtype::create(settings.executor->get_master(),
331                             gko::dim<2>(2 * this->metadata.local_size_x, 1));
332     // An initial guess.
333     std::shared_ptr<vec_vtype> init_guess = vec_vtype::create(
334         settings.executor, gko::dim<2>(this->metadata.local_size_x, 1));
335     init_guess->copy_from(local_rhs.get());
336
337     // std::vector<IndexType> local_converged_iter_count;
338
339     // Setup the windows for the onesided communication.
340     this->setup_windows(this->settings, this->metadata, global_solution);
341
342     const auto solver_settings =
343         (Settings::local_solver_settings::direct_solver_cholmod |
344          Settings::local_solver_settings::direct_solver_umfpack |
345          Settings::local_solver_settings::direct_solver_ginkgo |
346          Settings::local_solver_settings::iterative_solver_dealii |
347          Settings::local_solver_settings::iterative_solver_ginkgo) &
348         settings.local_solver;
349
350     ValueType local_residual_norm = -1.0, local_residual_norm0 = -1.0,
351             global_residual_norm = 0.0, global_residual_norm0 = -1.0;
352     metadata.iter_count = 0;
353     auto start_time = std::chrono::steady_clock::now();
354     int num_converged_procs = 0;
355
356     for (; metadata.iter_count < metadata.max_iters; ++(metadata.iter_count)) {
357         // Exchange the boundary values. The communication part.
358         MEASURE_ELAPSED_FUNC_TIME(
359             this->exchange_boundary(settings, metadata, global_solution), 0,
360             metadata.my_rank, boundary_exchange, metadata.iter_count);
361
362         // Update the boundary and interior values after the exchanging from
363         // other processes.
364         MEASURE_ELAPSED_FUNC_TIME(
365             this->update_boundary(settings, metadata, this->
366 local_solution,
367                             this->local_rhs, global_solution,
368                             this->interface_matrix),
369             1, metadata.my_rank, boundary_update, metadata.iter_count);
370
371         // Check for the convergence of the solver.
372         // num_converged_procs = 0;
373         MEASURE_ELAPSED_FUNC_TIME(
374             (Solve<ValueType, IndexType>::check_convergence(
375                 settings, metadata, this->comm_struct, this->convergence_vector,
376                 global_solution, this->local_solution, this->
377 local_matrix,
378                 work_vector, local_residual_norm, local_residual_norm0,
379                 global_residual_norm, global_residual_norm0,
380                 num_converged_procs)),
381             2, metadata.my_rank, convergence_check, metadata.iter_count);
382
383         // break if the solution diverges.
384         if (std::isnan(global_residual_norm) || global_residual_norm > 1e12) {
385             std::cout << " Rank " << metadata.my_rank << " diverged in "
386                 << metadata.iter_count << " iters " << std::endl;
387             std::exit(-1);
388         }
389
390         // break if all processes detect that all other processes have
391         // converged otherwise continue iterations.
392         if (num_converged_procs == metadata.num_subdomains) {
393             break;
394         } else {
395             MEASURE_ELAPSED_FUNC_TIME(
396                 (Solve<ValueType, IndexType>::local_solve(
397                     settings, metadata, this->local_matrix,
398                     this->triangular_factor_l, this->
399 triangular_factor_u,
400                     this->local_perm, this->local_inv_perm,
401                     work_vector,
402                     init_guess, this->local_solution)),

```

```

399         3, metadata.my_rank, local_solve, metadata.iter_count);
400         // Gather the local vector into the locally global vector for
401         // communication.
402         MEASURE_ELAPSED_FUNC_TIME (
403             (Communicate<ValueType, IndexType>::local_to_global_vector
404                 (
405                     settings, metadata, this->local_solution, global_solution)),
406                 4, metadata.my_rank, expand_local_vec, metadata.iter_count);
407     }
408     MPI_Barrier(MPI_COMM_WORLD);
409     auto elapsed_time = std::chrono::duration<ValueType>(
410         std::chrono::steady_clock::now() - start_time);
411     std::cout << " Rank " << metadata.my_rank << " converged in "
412         << metadata.iter_count << " iters " << std::endl;
413     ValueType mat_norm = -1.0, rhs_norm = -1.0, sol_norm = -1.0,
414         residual_norm = -1.0;
415     // Write the residuals and iterations to files
416     if (settings.write_iters_and_residuals &&
417         solver_settings ==
418             Settings::local_solver_settings::iterative_solver_ginkgo) {
419         std::string rank_string = std::to_string(metadata.my_rank);
420         if (metadata.my_rank < 10) {
421             rank_string = "0" + std::to_string(metadata.my_rank);
422         }
423         std::string filename = "iter_res_" + rank_string + ".csv";
424         write_iters_and_residuals(
425             metadata.num_subdomains, metadata.my_rank,
426             metadata.post_process_data.local_residual_vector_out.size(),
427             metadata.post_process_data.local_residual_vector_out,
428             metadata.post_process_data.local_converged_iter_count,
429             metadata.post_process_data.local_converged_resnorm, filename);
430     }
431
432     // Compute the final residual norm. Also gathers the solution from all
433     // subdomains.
434     Solve<ValueType, IndexType>::compute_residual_norm(
435         settings, metadata, global_matrix, global_rhs, global_solution,
436         mat_norm, rhs_norm, sol_norm, residual_norm);
437     gather_comm_data<ValueType, IndexType>(
438         metadata.num_subdomains, this->comm_struct, metadata.comm_data_struct);
439     // clang-format off
440     if (metadata.my_rank == 0)
441     {
442         std::cout
443             << " residual norm " << residual_norm << "\n"
444             << " relative residual norm of solution " << residual_norm/rhs_norm << "\n"
445             << " Time taken for solve " << elapsed_time.count()
446             << std::endl;
447         if (num_converged_procs < metadata.num_subdomains)
448         {
449             std::cout << " Did not converge in " << metadata.iter_count
450                 << " iterations."
451                 << std::endl;
452         }
453     }
454     // clang-format on
455     if (metadata.my_rank == 0) {
456         solution->copy_from(global_solution.get());
457     }
458
459     // Communicate<ValueType, IndexType>::clear(settings);
460 }

```

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

- schwarz\_base.hpp (394983b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/schwarz\_base.cpp (394983b)

## 7.14 schwz::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.*
- std::string [matrix\\_filename](#) = "null"  
*The string that contains the matrix file name to read from .*
- 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 [non\\_symmetric\\_matrix](#) = false  
*Is the matrix non-symmetric ? , Use GMRES for local solves.*
- unsigned int [restart\\_iter](#) = 1u  
*The restart iter for the GMRES solver.*
- 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\\_iters\\_and\\_residuals](#) = false  
*Enable writing the iters and residuals to a 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 factorization` = "cholmod"  
*The factorization for the local direct solver.*
- `std::string reorder`  
*The reordering for the local solve.*

### 7.14.1 Detailed Description

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

### 7.14.2 Member Data Documentation

#### 7.14.2.1 explicit\_laplacian

```
bool schwz::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 `schwz::SchwarzBase< ValueType, IndexType >::initialize()`.

#### 7.14.2.2 naturally\_ordered\_factor

```
bool schwz::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` (394983b)

## 7.15 `schwz::Solve< ValueType, IndexType >` Class Template Reference

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

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

### Additional Inherited Members

#### 7.15.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class schwz::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 (394983b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/solve.cpp (394983b)

## 7.16 schwz::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](#)) 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 >> &[global\\_solution](#)) 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 >> &[global\\_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

## 7.16.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>
class schwz::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.

## 7.16.2 Constructor &amp; Destructor Documentation

## 7.16.2.1 SolverRAS()

```
template<typename ValueType , typename IndexType >
schwz::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.

```
51      : SchwarzBase<ValueType, IndexType>(settings, metadata)
52 {}
```

## 7.16.3 Member Function Documentation

## 7.16.3.1 exchange\_boundary()

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

Exchanges the elements of the solution vector.

## Parameters

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



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

References [schwz::SchwarzBase< ValueType, IndexType >::cpu\\_work\\_vector](#), [schwz::Settings::comm\\_settings](#), [schwz::SchwarzBase< ValueType, IndexType >::enable\\_onesided](#), [schwz::SchwarzBase< ValueType, IndexType >::global\\_solution](#), and [schwz::SchwarzBase< ValueType, IndexType >::work\\_vector](#).

```

872 {
873     if (settings.comm_settings.enable_onesided) {
874         exchange_boundary_onesided<ValueType, IndexType>(
875             settings, metadata, this->comm_struct, this->work_vector,
876             this->cpu_work_vector, global_solution);
877     } else {
878         exchange_boundary_twosided<ValueType, IndexType>(
879             settings, metadata, this->comm_struct, global_solution);
880     }
881 }
```

### 7.16.3.2 setup\_local\_matrices()

```

template<typename ValueType , typename IndexType >
void schwz::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 )
[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 [schwz::Initialize< ValueType, IndexType >](#).

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

```

62 {
63     using mtx = gko::matrix::Csr<ValueType, IndexType>;
64     using vec_type = gko::Array<IndexType>;
65     using perm_type = gko::matrix::Permutation<IndexType>;
66     using arr = gko::Array<IndexType>;
67     auto my_rank = metadata.my_rank;
68     auto comm_size = metadata.comm_size;
69     auto num_subdomains = metadata.num_subdomains;
70     auto global_size = metadata.global_size;
71     auto mpi_itype = boost::mpi::get_mpi_datatype(*partition_indices.data());
72
73     MPI_Bcast(partition_indices.data(), global_size, mpi_itype, 0,
74             MPI_COMM_WORLD);
75
76     std::vector<IndexType> local_p_size(num_subdomains);
77     auto global_to_local = metadata.global_to_local->get_data();
78     auto local_to_global = metadata.local_to_global->get_data();
79
80     auto first_row = metadata.first_row->get_data();
81     auto permutation = metadata.permutation->get_data();
82     auto i_permutation = metadata.i_permutation->get_data();
83
84     auto nb = (global_size + num_subdomains - 1) /
num_subdomains;
85     auto partition_settings =
86     (Settings::partition_settings::partition_zoltan |
87     Settings::partition_settings::partition_metis |
88     Settings::partition_settings::partition_regular |
89     Settings::partition_settings::partition_regular2d |
90     Settings::partition_settings::partition_custom) &
91     settings.partition;
92
93     IndexType *gmat_row_ptrs = global_matrix->get_row_ptrs();
94     IndexType *gmat_col_idxs = global_matrix->get_col_idxs();
95     ValueType *gmat_values = global_matrix->get_values();
96
97     // default local p size set for 1 subdomain.
98     first_row[0] = 0;
99     for (auto p = 0; p < num_subdomains; ++p) {
100         local_p_size[p] = std::min(global_size - first_row[p], nb);
101         first_row[p + 1] = first_row[p] + local_p_size[p];
102     }
103
104     if (partition_settings == Settings::partition_settings::partition_metis ||
105         partition_settings ==
106         Settings::partition_settings::partition_regular2d) {
107         if (num_subdomains > 1) {
108             for (auto p = 0; p < num_subdomains; p++) {
109                 local_p_size[p] = 0;
110             }
111             for (auto i = 0; i < global_size; i++) {
112                 local_p_size[partition_indices[i]]++;
113             }
114             first_row[0] = 0;
115             for (auto p = 0; p < num_subdomains; ++p) {
116                 first_row[p + 1] = first_row[p] + local_p_size[p];
117             }
118             // permutation
119             for (auto i = 0; i < global_size; i++) {
120                 permutation[first_row[partition_indices[i]]] = i;
121                 first_row[partition_indices[i]]++;
122             }
123             for (auto p = num_subdomains; p > 0; p--) {
124                 first_row[p] = first_row[p - 1];
125             }
126             first_row[0] = 0;
127
128             // iperm
129             for (auto i = 0; i < global_size; i++) {
130                 i_permutation[permutation[i]] = i;
131             }
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_idxs()[nnz] =
145                 i_permutation[gmat_col_idxs[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
154
155     for (auto i = 0; i < global_size; i++) {
156         global_to_local[i] = 0;
157         local_to_global[i] = 0;
158     }
159     auto num = 0;
160     for (auto i = first_row[my_rank]; i < first_row[
my_rank + 1]; i++) {
161         global_to_local[i] = 1 + num;
162         local_to_global[num] = i;
163         num++;
164     }
165
166     IndexType old = 0;
167     for (auto k = 1; k < settings.overlap; k++) {
168         auto now = num;
169         for (auto i = old; i < now; i++) {
170             for (auto j = gmat_row_ptrs[local_to_global[i]];
171                  j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
172                 if (global_to_local[gmat_col_idxs[j]] == 0) {
173                     local_to_global[num] = gmat_col_idxs[j];
174                     global_to_local[gmat_col_idxs[j]] = 1 + num;
175                     num++;
176                 }
177             }
178         }
179         old = now;
180     }
181     metadata.local_size = local_p_size[my_rank];
182     metadata.local_size_x = num;
183     metadata.local_size_o = global_size;
184     auto local_size = metadata.local_size;
185     auto local_size_x = metadata.local_size_x;
186
187     metadata.overlap_size = num - metadata.local_size;
188     metadata.overlap_row = std::shared_ptr<vec_itype>(
189         new vec_itype(gko::Array<IndexType>::view(
190             settings.executor, metadata.overlap_size,
191             &(metadata.local_to_global->get_data()[metadata.local_size])),
192         std::default_delete<vec_itype>());
193
194     auto nnz_local = 0;
195     auto nnz_interface = 0;
196
197     for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
198         for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; j++) {
199             if (global_to_local[gmat_col_idxs[j]] != 0) {
200                 nnz_local++;
201             } else {
202                 std::cout << " debug: invalid edge?" << std::endl;
203             }
204         }
205     }
206     auto temp = 0;
207     for (auto k = 0; k < metadata.overlap_size; k++) {
208         temp = metadata.overlap_row->get_data()[k];
209         for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1]; j++) {
210             if (global_to_local[gmat_col_idxs[j]] != 0) {
211                 nnz_local++;
212             } else {
213                 nnz_interface++;
214             }
215         }
216     }
217
218     std::shared_ptr<mtx> local_matrix_compute;
219     local_matrix_compute = mtx::create(settings.executor->get_master(),
220                                       gko::dim<2>(local_size_x), nnz_local);
221     IndexType *lmat_row_ptrs = local_matrix_compute->get_row_ptrs();
222     IndexType *lmat_col_idxs = local_matrix_compute->get_col_idxs();
223     ValueType *lmat_values = local_matrix_compute->get_values();
224
225     std::shared_ptr<mtx> interface_matrix_compute;
226     if (nnz_interface > 0) {
227         interface_matrix_compute =
228             mtx::create(settings.executor->get_master(),
229                       gko::dim<2>(local_size_x), nnz_interface);
230     } else {
231         interface_matrix_compute = mtx::create(settings.executor->get_master());
232     }
233

```

```

234     IndexType *imat_row_ptrs = interface_matrix_compute->get_row_ptrs();
235     IndexType *imat_col_idxs = interface_matrix_compute->get_col_idxs();
236     ValueType *imat_values = interface_matrix_compute->get_values();
237
238     num = 0;
239     nnz_local = 0;
240     auto nnz_interface_temp = 0;
241     lmat_row_ptrs[0] = nnz_local;
242     if (nnz_interface > 0) {
243         imat_row_ptrs[0] = nnz_interface_temp;
244     }
245     // Local interior matrix
246     for (auto i = first_row[my_rank]; i < first_row[my_rank + 1]; ++i) {
247         for (auto j = gmat_row_ptrs[i]; j < gmat_row_ptrs[i + 1]; ++j) {
248             if (global_to_local[gmat_col_idxs[j]] != 0) {
249                 lmat_col_idxs[nnz_local] =
250                     global_to_local[gmat_col_idxs[j]] - 1;
251                 lmat_values[nnz_local] = gmat_values[j];
252                 nnz_local++;
253             }
254         }
255         if (nnz_interface > 0) {
256             imat_row_ptrs[num + 1] = nnz_interface_temp;
257         }
258         lmat_row_ptrs[num + 1] = nnz_local;
259         num++;
260     }
261
262     // Interface matrix
263     if (nnz_interface > 0) {
264         nnz_interface = 0;
265         for (auto k = 0; k < metadata.overlap_size; k++) {
266             temp = metadata.overlap_row->get_data()[k];
267             for (auto j = gmat_row_ptrs[temp]; j < gmat_row_ptrs[temp + 1];
268                 j++) {
269                 if (global_to_local[gmat_col_idxs[j]] != 0) {
270                     lmat_col_idxs[nnz_local] =
271                         global_to_local[gmat_col_idxs[j]] - 1;
272                     lmat_values[nnz_local] = gmat_values[j];
273                     nnz_local++;
274                 } else {
275                     imat_col_idxs[nnz_interface] = gmat_col_idxs[j];
276                     imat_values[nnz_interface] = gmat_values[j];
277                     nnz_interface++;
278                 }
279             }
280             lmat_row_ptrs[num + 1] = nnz_local;
281             imat_row_ptrs[num + 1] = nnz_interface;
282             num++;
283         }
284     }
285     auto now = num;
286     for (auto i = old; i < now; i++) {
287         for (auto j = gmat_row_ptrs[local_to_global[i]];
288             j < gmat_row_ptrs[local_to_global[i] + 1]; j++) {
289             if (global_to_local[gmat_col_idxs[j]] == 0) {
290                 local_to_global[num] = gmat_col_idxs[j];
291                 global_to_local[gmat_col_idxs[j]] = 1 + num;
292                 num++;
293             }
294         }
295     }
296
297     local_matrix = mtx::create(settings.executor);
298     local_matrix->copy_from(gko::lend(local_matrix_compute));
299     interface_matrix = mtx::create(settings.executor);
300     interface_matrix->copy_from(gko::lend(interface_matrix_compute));
301
302     local_matrix->sort_by_column_index();
303     interface_matrix->sort_by_column_index();
304 }

```

### 7.16.3.3 setup\_windows()

```

template<typename ValueType , typename IndexType >
void schwz::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 [schwz::Communicate< ValueType, IndexType >](#).

References [schwz::Communicate< ValueType, IndexType >::comm\\_struct::cpu\\_rcv\\_buffer](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::cpu\\_send\\_buffer](#), [schwz::SchwarzBase< ValueType, IndexType >::cpu\\_work\\_vector](#), [schwz::Settings::comm\\_settings::enable\\_get](#), [schwz::Settings::comm\\_settings::enable\\_lock\\_all](#), [schwz::Settings::comm\\_settings::enable\\_one\\_by\\_one](#), [schwz::Settings::comm\\_settings::enable\\_onesided](#), [schwz::Settings::comm\\_settings::enable\\_overlap](#), [schwz::Settings::comm\\_settings::enable\\_put](#), [schwz::Settings::executor](#), [schwz::Settings::executor\\_string](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::get\\_displacements](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::get\\_request](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::global\\_get](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::global\\_put](#), [schwz::SchwarzBase< ValueType, IndexType >::global\\_solution](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::is\\_local\\_neighbor](#), [schwz::Metadata< ValueType, IndexType >::iter\\_count](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::local\\_get](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::local\\_neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::local\\_neighbors\\_out](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::local\\_num\\_neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::local\\_num\\_neighbors\\_out](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::local\\_put](#), [schwz::Metadata< ValueType, IndexType >::local\\_size\\_o](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::neighbors\\_out](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::num\\_neighbors\\_in](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::num\\_neighbors\\_out](#), [schwz::Metadata< ValueType, IndexType >::num\\_subdomains](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::put\\_displacements](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::put\\_request](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::rcv\\_buffer](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::remote\\_get](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::remote\\_put](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::send\\_buffer](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::window\\_cpu\\_rcv\\_buffer](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::window\\_cpu\\_send\\_buffer](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::window\\_rcv\\_buffer](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::window\\_send\\_buffer](#), [schwz::Communicate< ValueType, IndexType >::comm\\_struct::window\\_x](#), and [schwz::SchwarzBase< ValueType, IndexType >::work\\_vector](#).

```
521 {
522     using vec_itype = gko::Array<IndexType>;
523     using vec_vtype = gko::matrix::Dense<ValueType>;
524     auto num_subdomains = metadata.num_subdomains;
525     auto local_size_o = metadata.local_size_o;
526     auto neighbors_in = this->comm_struct.neighbors_in->get_data();
527     auto global_get = this->comm_struct.global_get->get_data();
528     auto neighbors_out = this->comm_struct.neighbors_out->get_data();
529     auto global_put = this->comm_struct.global_put->get_data();
530
531     // set displacement for the MPI buffer
532     auto get_displacements = this->comm_struct.get_displacements->get_data();
533     auto put_displacements = this->comm_struct.put_displacements->get_data();
534     {
535         std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
536         tmp_num_comm_elems[0] = 0;
537         for (auto j = 0; j < this->comm_struct.num_neighbors_in; j++) {
538             if ((global_get[j])[0] > 0) {
539                 int p = neighbors_in[j];
540                 tmp_num_comm_elems[p + 1] = (global_get[j])[0];
541             }
542         }
543     }
544 }
```

```

541     }
542 }
543 for (auto j = 0; j < num_subdomains; j++) {
544     tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
545 }
546
547 auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
548 MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, put_displacements,
549             1, mpi_itype, MPI_COMM_WORLD);
550 }
551
552 {
553     std::vector<IndexType> tmp_num_comm_elems(num_subdomains + 1, 0);
554     tmp_num_comm_elems[0] = 0;
555     for (auto j = 0; j < this->comm_struct.num_neighbors_out; j++) {
556         if ((global_put[j])[0] > 0) {
557             int p = neighbors_out[j];
558             tmp_num_comm_elems[p + 1] = (global_put[j])[0];
559         }
560     }
561     for (auto j = 0; j < num_subdomains; j++) {
562         tmp_num_comm_elems[j + 1] += tmp_num_comm_elems[j];
563     }
564
565     auto mpi_itype = boost::mpi::get_mpi_datatype(tmp_num_comm_elems[0]);
566     MPI_Alltoall(tmp_num_comm_elems.data(), 1, mpi_itype, get_displacements,
567                 1, mpi_itype, MPI_COMM_WORLD);
568 }
569
570 // setup windows
571 if (settings.comm_settings.enable_onesided) {
572     // Onesided
573     MPI_Win_create(main_buffer->get_values(),
574                   main_buffer->get_size()[0] * sizeof(ValueType),
575                   sizeof(ValueType), MPI_INFO_NULL, MPI_COMM_WORLD,
576                   &(this->comm_struct.window_x));
577 }
578
579
580 if (settings.comm_settings.enable_onesided) {
581     // MPI_Alloc_mem ? Custom allocator ? TODO
582     MPI_Win_create(this->local_residual_vector->get_values(),
583                   (num_subdomains) * sizeof(ValueType), sizeof(ValueType),
584                   MPI_INFO_NULL, MPI_COMM_WORLD,
585                   &(this->window_residual_vector));
586     std::vector<IndexType> zero_vec(num_subdomains, 0);
587     gko::Array<IndexType> temp_array(settings.executor->get_master(),
588                                     zero_vec.begin(), zero_vec.end());
589     this->convergence_vector = std::shared_ptr<vec_itype>(
590         new vec_itype(settings.executor->get_master(), temp_array),
591         std::default_delete<vec_itype>());
592     this->convergence_sent = std::shared_ptr<vec_itype>(
593         new vec_itype(settings.executor->get_master(), num_subdomains),
594         std::default_delete<vec_itype>());
595     this->convergence_local = std::shared_ptr<vec_itype>(
596         new vec_itype(settings.executor->get_master(), num_subdomains),
597         std::default_delete<vec_itype>());
598     MPI_Win_create(this->convergence_vector->get_data(),
599                   (num_subdomains) * sizeof(IndexType), sizeof(IndexType),
600                   MPI_INFO_NULL, MPI_COMM_WORLD,
601                   &(this->window_convergence));
602 }
603
604 if (settings.comm_settings.enable_onesided && num_subdomains > 1) {
605     // Lock all windows.
606     if (settings.comm_settings.enable_get &&
607         settings.comm_settings.enable_lock_all) {
608         MPI_Win_lock_all(0, this->comm_struct.window_send_buffer);
609         MPI_Win_lock_all(0, this->comm_struct.window_cpu_send_buffer);
610     }
611     if (settings.comm_settings.enable_put &&
612         settings.comm_settings.enable_lock_all) {
613         MPI_Win_lock_all(0, this->comm_struct.window_recv_buffer);
614         MPI_Win_lock_all(0, this->comm_struct.window_cpu_recv_buffer);
615     }
616     if (settings.comm_settings.enable_one_by_one &&
617         settings.comm_settings.enable_lock_all) {
618         MPI_Win_lock_all(0, this->comm_struct.window_x);
619     }
620     MPI_Win_lock_all(0, this->window_residual_vector);
621     MPI_Win_lock_all(0, this->window_convergence);
622 }
623 }

```

## 7.16.3.4 update\_boundary()

```
template<typename ValueType , typename IndexType >
void schwz::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 >> & global_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.
<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>global_solution</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 [schwz::Communicate< ValueType, IndexType >](#).

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

```
892 {
893     using vec_vtype = gko::matrix::Dense<ValueType>;
894     auto one = gko::initialize<gko::matrix::Dense<ValueType>>(
895         {1.0}, settings.executor);
896     auto neg_one = gko::initialize<gko::matrix::Dense<ValueType>>(
897         {-1.0}, settings.executor);
898     auto local_size_x = metadata.local_size_x;
899     local_solution->copy_from(local_rhs.get());
900     if (metadata.num_subdomains > 1 && settings.overlap > 0) {
901         auto temp_solution = vec_vtype::create(
902             settings.executor, local_solution->get_size(),
903             gko::Array<ValueType>::view(settings.executor,
904                 local_solution->get_size()[0],
905                 global_solution->get_values()),
906             1);
907         interface_matrix->apply(neg_one.get(), temp_solution.get(), one.get(),
908             local_solution.get());
909     }
910 }
```

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

- [restricted\\_schwarz.hpp \(394983b\)](#)
- [/home/runner/work/schwarz-lib/schwarz-lib/source/restricted\\_schwarz.cpp \(394983b\)](#)

## 7.17 UmfpackError Class Reference

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

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

### Public Member Functions

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

#### 7.17.1 Detailed Description

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

#### 7.17.2 Constructor & Destructor Documentation

##### 7.17.2.1 UmfpackError()

```
UmfpackError::UmfpackError (
    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

```
205         : Error(file, line, func + ": " + get_error(error_code))
206     {}
```

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

- exception.hpp (394983b)
- /home/runner/work/schwarz-lib/schwarz-lib/source/exception.cpp (394983b)



## 7.18 schwz::Utils< ValueType, IndexType > Struct Template Reference

The utilities class which provides some checks and basic utilities.

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

### 7.18.1 Detailed Description

```
template<typename ValueType = gko::default_precision, typename IndexType = gko::int32>  
struct schwz::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` (394983b)
- `/home/runner/work/schwarz-lib/schwarz-lib/source/utils.cpp` (394983b)



# Index

- BadDimension, [19](#)
  - BadDimension, [19](#)
- Communicate, [9](#)
- CudaError, [28](#)
  - CudaError, [29](#)
- CusparsError, [29](#)
  - CusparsError, [30](#)
- exchange\_boundary
  - schwz::Communicate, [26](#)
  - schwz::SolverRAS, [50](#)
- explicit\_laplacian
  - schwz::Settings, [48](#)
- generate\_rhs
  - schwz::Initialize, [32](#)
- global\_get
  - schwz::Communicate::comm\_struct, [22](#)
- global\_put
  - schwz::Communicate::comm\_struct, [23](#)
- Initialization, [10](#)
- is\_local\_neighbor
  - schwz::Communicate::comm\_struct, [23](#)
- local\_get
  - schwz::Communicate::comm\_struct, [23](#)
- local\_put
  - schwz::Communicate::comm\_struct, [23](#)
- local\_solver\_tolerance
  - schwz::Metadata, [38](#)
- local\_to\_global\_vector
  - schwz::Communicate, [26](#)
- MetisError, [38](#)
  - MetisError, [39](#)
- naturally\_ordered\_factor
  - schwz::Settings, [48](#)
- partition
  - schwz::Initialize, [32](#)
- print\_matrix
  - schwz::SchwarzBase, [42](#)
- print\_vector
  - schwz::SchwarzBase, [44](#)
- ProcessTopology, [15](#)
- remote\_get
  - schwz::Communicate::comm\_struct, [24](#)
- remote\_put
  - schwz::Communicate::comm\_struct, [24](#)
- run
  - schwz::SchwarzBase, [44](#)
- Schwarz Class, [11](#)
- SchwarzBase
  - schwz::SchwarzBase, [41](#)
- schwz, [15](#)
- schwz::CommHelpers, [16](#)
- schwz::Communicate
  - exchange\_boundary, [26](#)
  - local\_to\_global\_vector, [26](#)
  - setup\_windows, [27](#)
  - update\_boundary, [27](#)
- schwz::Communicate< ValueType, IndexType >, [25](#)
- schwz::Communicate< ValueType, IndexType >↔
  - ::comm\_struct, [21](#)
- schwz::Communicate::comm\_struct
  - global\_get, [22](#)
  - global\_put, [23](#)
  - is\_local\_neighbor, [23](#)
  - local\_get, [23](#)
  - local\_put, [23](#)
  - remote\_get, [24](#)
  - remote\_put, [24](#)
- schwz::Initialize
  - generate\_rhs, [32](#)
  - partition, [32](#)
  - setup\_global\_matrix, [33](#)
  - setup\_local\_matrices, [34](#)
  - setup\_vectors, [35](#)
- schwz::Initialize< ValueType, IndexType >, [31](#)
- schwz::Metadata
  - local\_solver\_tolerance, [38](#)
  - tolerance, [38](#)
- schwz::Metadata< ValueType, IndexType >, [36](#)
- schwz::Metadata< ValueType, IndexType >::post\_↔
  - process\_data, [39](#)
- schwz::PartitionTools, [17](#)
- schwz::SchwarzBase
  - print\_matrix, [42](#)
  - print\_vector, [44](#)
  - run, [44](#)
  - SchwarzBase, [41](#)
- schwz::SchwarzBase< ValueType, IndexType >, [40](#)
- schwz::Settings, [46](#)
  - explicit\_laplacian, [48](#)
  - naturally\_ordered\_factor, [48](#)
- schwz::Settings::comm\_settings, [20](#)

- schwz::Settings::convergence\_settings, [28](#)
- schwz::Solve< ValueType, IndexType >, [48](#)
- schwz::SolverRAS< ValueType, IndexType >, [49](#)
- schwz::SolverRAS
  - exchange\_boundary, [50](#)
  - setup\_local\_matrices, [51](#)
  - setup\_windows, [54](#)
  - SolverRAS, [50](#)
  - update\_boundary, [56](#)
- schwz::SolverTools, [17](#)
- schwz::Utils< ValueType, IndexType >, [59](#)
- schwz::conv\_tools, [16](#)
- schwz::device\_guard, [30](#)
- setup\_global\_matrix
  - schwz::Initialize, [33](#)
- setup\_local\_matrices
  - schwz::Initialize, [34](#)
  - schwz::SolverRAS, [51](#)
- setup\_vectors
  - schwz::Initialize, [35](#)
- setup\_windows
  - schwz::Communicate, [27](#)
  - schwz::SolverRAS, [54](#)
- Solve, [12](#)
- SolverRAS
  - schwz::SolverRAS, [50](#)
- tolerance
  - schwz::Metadata, [38](#)
- UmfpackError, [58](#)
  - UmfpackError, [58](#)
- update\_boundary
  - schwz::Communicate, [27](#)
  - schwz::SolverRAS, [56](#)
- Utils, [13](#)