Repast HPC

2.0

Generated by Doxygen 1.8.4

Sun Aug 11 2013 12:21:30

Contents

Chapter 1

Repast HPC: A High-Performance Agent-Based Modeling Platform

By Argonne National Laboratory, 2009-2013

1.1 What is Repast HPC?

Repast HPC is an Agent-Based Modeling Platform in the spirit of Repast Simphony but designed for top-500 high-performance computing systems (supercomputers).

Chapter 2

Deprecated List

Class repast::CellContents< AgentContent, GPType >
Replaced by ProjectionInfoPacket as of Version 2.0

Class repast::EdgeExporter< E >
As of Version 2.0 replaced by ProjectionInfoPacket

Class repast::ExportRequest
As of Version 2.0

Class repast::GridBufferSyncher< T, GPType >
As of Version 2.0

Class repast::ItemReceipt< E >
As of Version 2.0 replaced by ProjectionInfoPacket objects.

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

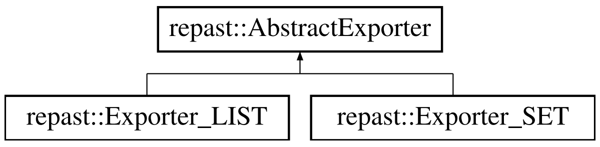
Class Documentation

5.1 repast::AbstractExporter Class Reference

Responsible for keeping a list of the agents that have been requested by other processes for which data is to be sent when agents' states are synchronized, and for packaging and sending that data during synchronization.

```
#include <AgentImporterExporter.h>
```

Inheritance diagram for repast::AbstractExporter:



Public Types

- typedef std::map$<$ int,
  std::set$<$ AgentStatus $>$ $>$ **StatusMap**

Public Member Functions

- virtual void registerIncomingRequests (std::vector$<$ AgentRequest $>$ &requests)=0

  Makes a record of the data receives (in the form of a vector of AgentRequests) so that the agents' data can be sent to the requesting processes.
- virtual void incorporateAgentExporterInfo (std::map$<$ int, AgentRequest $*$ $>$ info)

  The set of information received here comprises the information that some other process was using to export information about agents that are now being moved to this process.
- virtual void agentRemoved (const AgentId &id)

  1) Removes the agent export information from this process 2) Updates the outgoing status change buffer to include the status change for this agent to all procs to which this agent was being exported (except if one of these was the proc to which the agent is now moving; this is omitted)
- virtual void agentMoved (const AgentId &id, int process)

  1) Removes the agent export information from this process 2) Places a copy of the agent export information into the outgoing buffer 3) Updates the outgoing status change buffer to include the status change for this agent to all procs to which this agent was being exported (except if one of these was the proc to which the agent is now moving; this is omitted)
- virtual const std::set$<$ int $>$ & getProcessesExportedTo ()

  Gets the list of processes this exporter is sending information to.

- AgentExporterInfo $*$ getAgentExportInfo (int destProc)

  Gets the export information that has been placed into the 'outgoing agent export information' buffer because agents that were being exported are being sent to a new process, for the specified process.

- const StatusMap $*$ getOutgoingStatusChanges ()

  Gets the set of status changes for the exported agents.

- void clearAgentExportInfo ()

  Clears the outgoing agent export information buffer; should be called after the information is sent.

- void clearStatusMap ()

  Clears the outgoing status information buffer; should be called after the information is sent.

- virtual const std::map$<$ int,
  AgentRequest $>$ & getAgentsToExport ()

  Gets the list of agents being exported by this exported, as a map by ints representing the processes to which information will be sent.

- **AbstractExporter** (StatusMap $*$outgoingStatusMap, AgentExporterData $*$outgoingAgentExporterInfo)
- virtual std::string getReport ()=0

  Gets a printable report of the state of this object.

- virtual void **clear** ()
- virtual void **clearExportToSpecificProc** (int rank)

Protected Attributes

- StatusMap $*$ **outgoingStatusChanges**
- AgentExporterData $*$ **outgoingAgentExporterInformation**
- std::set$<$ int $>$ **processesExportedTo**
- std::map$<$ int, AgentRequest $>$ **exportedMap**

5.1.1 Detailed Description

Responsible for keeping a list of the agents that have been requested by other processes for which data is to be sent when agents' states are synchronized, and for packaging and sending that data during synchronization.

It is also responsible for exchanging this 'export' information when any of the agents that it is exporting are being moved to other processes; when an agent moves, its new home process must be able to assume the same export duties that its original process was performing.

5.1.2 Member Function Documentation

5.1.2.1 void AbstractExporter::incorporateAgentExporterInfo (std::map$<$ int, AgentRequest $*$ $>$ *info*) [virtual]

The set of information received here comprises the information that some other process was using to export information about agents that are now being moved to this process.

This method takes that information and incorporates it into this exporter, so that this exporter can now export the agents' information to the processes that have requested it.

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

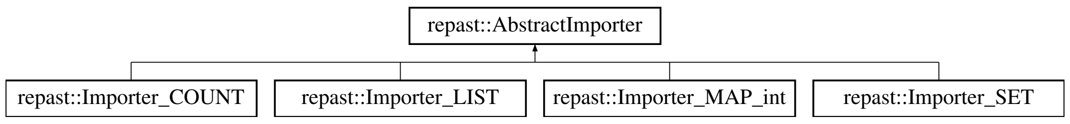
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.2 repast::AbstractImporter Class Reference

This class manages importing agent information; primarily this means constructing the appropriate mpi receives when agent information is to be exchanged.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::AbstractImporter:



Public Member Functions

- virtual const std::set< int > & getExportingProcesses ()

  Gets a const reference to the set of ints representing the processes that are sending this process agent information.
- virtual void registerOutgoingRequests (AgentRequest &req)=0

  Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.
- virtual void importedAgentIsRemoved (const AgentId &id)=0

  Notifies this importer that the agent that it (presumably) has been importing has been removed from the simulation on its home process, and the information for that agent will no longer be sent.
- virtual void importedAgentIsMoved (const AgentId &id, int newProcess)=0

  Notifies this importer that the agent that it (presumably) has been importing from another process has been moved; its information will now be received from its new home process (unless the agent was moved to this process)
- void importedAgentIsNowLocal (const AgentId &id)

  Some semantic sugar; operationally this is the same as 'importedAgentIsRemoved'.
- virtual std::string getReport ()=0

  Get a printable indication of the data in this object.
- virtual void **getSetOfAgentsBeingImported** (std::set< AgentId > &set)=0
- virtual void **clear** ()

Protected Attributes

- std::set< int > **exportingProcesses**

5.2.1 Detailed Description

This class manages importing agent information; primarily this means constructing the appropriate mpi receives when agent information is to be exchanged.

However, this class can also define specific semantics that can apply to agent requests- what to do in the case that an agent is requested twice, for example.

5.2.2 Member Function Documentation

5.2.2.1 virtual void repast::AbstractImporter::registerOutgoingRequests (**AgentRequest** & *req*) `[pure virtual]`

Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.

The record must at a minimum indicate which other processes are sending agent information, but may include other information, such as how many times a particular agent has been requested.

Implemented in repast::Importer_MAP_int, repast::Importer_SET, repast::Importer_LIST, and repast::Importer_C-OUNT.

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

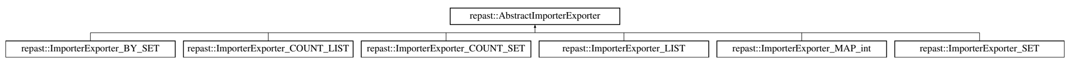
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.3 repast::AbstractImporterExporter Class Reference

Wraps and Importer and an Exporter so that both use commensurate semantics and all imports and exports are balanced.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::AbstractImporterExporter:


repast::AbstractImporterExporter
repast::ImporterExporter_BY_SET
repast::ImporterExporter_COUNT_LIST
repast::ImporterExporter_COUNT_SET
repast::ImporterExporter_LIST
repast::ImporterExporter_MAP_int
repast::ImporterExporter_SET

Public Member Functions

- **AbstractImporterExporter** (AbstractImporter ∗i, AbstractExporter ∗e)
- virtual const std::set< int > & **getExportingProcesses** ()
- virtual void **registerOutgoingRequests** (AgentRequest &req)
- virtual void **importedAgentIsRemoved** (const AgentId &id)
- virtual void **importedAgentIsMoved** (const AgentId &id, int newProcess)
- virtual void **importedAgentIsNowLocal** (const AgentId &id)
- virtual void **getSetOfAgentsBeingImported** (std::set< AgentId > &set)
- virtual const AbstractExporter::StatusMap ∗ **getOutgoingStatusChanges** ()
- virtual const std::set< int > & **getProcessesExportedTo** ()
- virtual void **registerIncomingRequests** (std::vector< AgentRequest > &requests)
- virtual void **agentRemoved** (const AgentId &id)
- virtual void **agentMoved** (const AgentId &id, int process)
- virtual void **incorporateAgentExporterInfo** (std::map< int, AgentRequest ∗ > info)
- virtual void **clearStatusMap** ()
- virtual AgentExporterInfo ∗ **getAgentExportInfo** (int destProc)
- virtual void **clearAgentExportInfo** ()
- virtual const std::map< int, AgentRequest > & **getAgentsToExport** ()
- virtual void exchangeAgentStatusUpdates (boost::mpi::communicator world, std::vector< std::vector< Agent-Status > ∗ > &statusUpdates)

  Exchanges the contents of the 'statusMap' with the destination processes, updating the status (moved or removed) for all agents being exported.
- virtual std::string version ()=0

  Returns the version of this AbstractImporterExporter.
- virtual std::string getReport ()

  Gets a printable report of the state of this object.
- virtual void **clear** ()
- virtual void **clearExporter** ()
- virtual void **clearExportToSpecificProc** (int rank)

Protected Attributes

- AbstractImporter ∗ **importer**
- AbstractExporter ∗ **exporter**

5.3.1 Detailed Description

Wraps and Importer and an Exporter so that both use commensurate semantics and all imports and exports are balanced.

Most methods are pass-through to the underlying importer or exporter.

5.3.2 Member Function Documentation

5.3.2.1 void AbstractImporterExporter::exchangeAgentStatusUpdates (boost::mpi::communicator *world,* std::vector< std::vector< **AgentStatus** > ∗ > & *statusUpdates*) `[virtual]`

Exchanges the contents of the 'statusMap' with the destination processes, updating the status (moved or removed) for all agents being exported.

Returns this information in the statusUpdates vector.

5.3.2.2 virtual std::string repast::AbstractImporterExporter::version ( ) `[pure virtual]`

Returns the version of this AbstractImporterExporter.

The version is a string that indicates the semantic version of the importer and the exporter (e.g. "COUNT_LIST")

Implemented in repast::ImporterExporter_BY_SET, repast::ImporterExporter_MAP_int, repast::ImporterExporter_-SET, repast::ImporterExporter_LIST, repast::ImporterExporter_COUNT_SET, and repast::ImporterExporter_COU-NT_LIST.

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

- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.4 repast::Agent Class Reference

Interface for agent classes.

`#include <AgentId.h>`

Public Member Functions

- virtual AgentId & getId ()=0

  Gets the AgentId for this Agent.
- virtual const AgentId & getId () const =0

  Gets the AgentId for this Agent.

5.4.1 Detailed Description

Interface for agent classes.

5.4.2 Member Function Documentation

5.4.2.1 virtual **AgentId&** repast::Agent::getId ( ) `[pure virtual]`

Gets the AgentId for this Agent.

Returns

the AgentId for this Agent.

5.4.2.2 virtual const **AgentId&** repast::Agent::getId ( ) const `[pure virtual]`

Gets the AgentId for this Agent.

Returns

the AgentId for this Agent.

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

- repast_hpc/AgentId.h

5.5 repast::AgentExporterData Class Reference

Data structure for exporter data that is to be sent to other processes when the agents being exported are moved.

`#include <AgentImporterExporter.h>`

Public Member Functions

- void addData (const AgentId &id, const int destProc, const int sourceProc, const int numberOfCopies=1)

  Adds an agent ID to this list of data that is being exported to a specific processor (destProc), so that the agent's information will be exported to another processor (sourceProc).
- AgentExporterInfo ∗ dataForProc (int destProc)

  Gets the packaged set of information to be sent to a specific processor.
- void clear ()

  Clears this data structure.
- void removeAllDataForAgent (AgentId &id)

  Remove all the data for a specific agent; useful when the agent is removed.
- void selectSet (std::string setName)

  Specifies that add and retrieve actions are to be performed on the subset of data identified by the given set name.

5.5.1 Detailed Description

Data structure for exporter data that is to be sent to other processes when the agents being exported are moved.

Note that the internal data structure is protected; classes can use this data without knowing its actual internal structure.

5.5.2 Member Function Documentation

5.5.2.1 void AgentExporterData::selectSet (std::string *setName*)

Specifies that add and retrieve actions are to be performed on the subset of data identified by the given set name. (Does not affect 'clear' or 'removeAllDataForAgent')

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

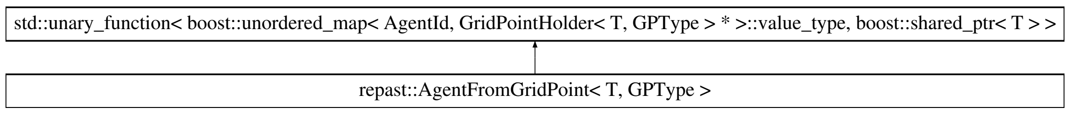
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.6 repast::AgentFromGridPoint< T, GPType > Struct Template Reference

Unary function used in the transform_iterator that allows context iterators to return the agent maps values.

```
#include <BaseGrid.h>
```

Inheritance diagram for repast::AgentFromGridPoint< T, GPType >:


std::unary_function< boost::unordered_map< AgentId, GridPointHolder< T, GPType > * >::value_type, boost::shared_ptr< T > >
repast::AgentFromGridPoint< T, GPType >

Public Member Functions

- boost::shared_ptr< T > **operator()** (const typename boost::unordered_map< AgentId, GridPointHolder< T, GPType > ∗ >::value_type &value) const

5.6.1 Detailed Description

template<typename T, typename GPType>struct repast::AgentFromGridPoint< T, GPType >

Unary function used in the transform_iterator that allows context iterators to return the agent maps values.

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

- repast_hpc/BaseGrid.h

5.7 repast::AgentHashId< AgentType > Struct Template Reference

operator() implementation that returns the hashcode of an agent via its AgentId.

```
#include <AgentId.h>
```

Public Member Functions

- std::size_t **operator()** (const AgentType ∗agent) const

5.7.1 Detailed Description

template<typename AgentType>struct repast::AgentHashId< AgentType >

operator() implementation that returns the hashcode of an agent via its AgentId.

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

- repast_hpc/AgentId.h

5.8 repast::AgentId Class Reference

Agent identity information.

`#include <AgentId.h>`

Public Member Functions

- AgentId ()
  No-arg constructor necessary for serialization.
- AgentId (int id, int startProc, int agentType, int currentProc=-1)
  Creates an AgentId.
- int id () const
  Gets the id component of this AgentId.
- int startingRank () const
  Gets the starting rank component of this AgentId.
- int agentType () const
  Gets the agent type component of this AgentId.
- int currentRank () const
  Gets the current process rank of this AgentId.
- void currentRank (int val)
  Sets the current process rank of this AgentId.
- std::size_t hashcode () const
  Gets the hashcode for this AgentId.

Friends

- class **boost::serialization::access**
- std::ostream & operator<< (std::ostream &os, const AgentId &id)
  Writes the agent id to the ostream.
- bool operator== (const AgentId &one, const AgentId &two)
  Equality operator.
- bool operator< (const AgentId &one, const AgentId &two)
  A comparison operator for use with std::set.

5.8.1 Detailed Description

Agent identity information.

An Agent ID consists of four values: 1) a numerical identifier; 2) the process on which the agent was created; 3) a numerical identifier that indicates the agent's type (in simulation semantic terms, not a software object type); and 4) the process on which the agent is a local agent. Each agent should be uniquely identified by an AgentId using the first three of the four values, which should be immutable. The fourth value can change throughout the simulation.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 repast::AgentId::AgentId (int *id*, int *startProc*, int *agentType*, int *currentProc* = `-1`)

Creates an AgentId.

The combination of the first three parameters should uniquely identify the agent.

Parameters

id	the agent's id
startProc	the rank of the agent's starting process
agentType	the agent's type (user defined)
currentProc	the rank where the agent is a local agent

5.8.3 Member Function Documentation

5.8.3.1 int repast::AgentId::agentType ( ) const `[inline]`

Gets the agent type component of this AgentId.

Returns

the agent type component of this AgentId.

5.8.3.2 int repast::AgentId::currentRank ( ) const `[inline]`

Gets the current process rank of this AgentId.

The current rank identifies which process the agent with this AgentId is currently on.

Returns

the current process rank of this AgentId.

5.8.3.3 void repast::AgentId::currentRank (int *val*) `[inline]`

Sets the current process rank of this AgentId.

The current rank identifies which process the agent with this AgentId is currently on.

Parameters

val	the current process rank

5.8.3.4 std::size_t repast::AgentId::hashcode ( ) const `[inline]`

Gets the hashcode for this AgentId.

Returns

the hashcode for this AgentId.

5.8.3.5 int repast::AgentId::id ( ) const [inline]

Gets the id component of this AgentId.

Returns

the id component of this AgentId.

5.8.3.6 int repast::AgentId::startingRank ( ) const [inline]

Gets the starting rank component of this AgentId.

Returns

the starting rank component of this AgentId.

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

- repast_hpc/AgentId.h
- repast_hpc/AgentId.cpp

5.9 repast::AgentRequest Class Reference

Encapsulates a request made by one process for agents in another.

```
#include <AgentRequest.h>
```

Public Member Functions

- AgentRequest (int sourceProcess)

  Creates an AgentRequest that comes from the specified process.
- AgentRequest (int sourceProcess, int targetProcess)

  Creates an AgentRequest made from the source process to the target process.
- void addAll (const AgentRequest &req)

  Adds all the agent ids (both requests and cancellations) in req to this AgentRequest.
- void addAllRequests (const AgentRequest &req)

  Adds all the agent ids in req to this request, including only the ids that are requests and not those that are cancellations.
- void addAllCancellations (const AgentRequest &req)

  Adds all the agent ids in req to this request, including only the ids that are cancellations and not those that are requests.
- const std::vector< AgentId > & requestedAgents () const

  Gets a reference to the vector of requested agents.
- const std::vector< AgentId > & cancellations () const

  Gets a reference to the vector of cancellations.
- bool remove (const AgentId &id, bool removeAllInstances=true)

  Removes the specified id from the lists of requested agents, including both requests and cancellations.
- bool removeRequest (const AgentId &id, bool removeAllInstances=true)

  Removes the specified id from the list of agent requests; does not affect the list of cancellations.
- bool removeCancellation (const AgentId &id, bool removeAllInstances=true)

  Removes the specified id from the list of agent request cancellations; does not affect the list of requests.
- void targets (std::set< int > &targets)

Puts the targets of all the requests into the set.

- void targetsOfRequests (std::set< int > &targets)

  Puts the targets of all the requests into the set, including only the requests and not the cancellations.
- void targetsOfCancellations (std::set< int > &targets)

  Puts the targets of all the requests into the set, including only the requests and not the cancellations.
- void addRequest (const AgentId &id)

  Adds the specified agent to the collection agents being requested.
- void addCancellation (const AgentId &id)

  Adds the specified agent to the collection of agents for which a previous request is being cancelled.
- int requestCount () const

  Gets the number agents requested.
- int requestCountRequested () const

  Gets the number of agents requested, counting only the requests and not the cancellations.
- int requestCountCancellations () const

  Gets the number of agents requested, counting only the cancellations and not the requests.
- bool contains (const AgentId &id)

  Returns true if this AgentRequest contains a request for the specified id (either a request or a cancellation), otherwise false.
- bool containsInRequests (const AgentId &id)

  Returns true if the list of requests contains the specified id (the list of cancellations is ignored)
- bool containsInCancellations (const AgentId &id)

  Returns true if the list of cancellations contains the specified id (the list of requests is ignored)
- int sourceProcess () const

  Gets the source process of these requests, that is, the process making the request.
- int targetProcess () const

  If the requested agent ids are all on the same process then target process will identify that process.

Friends

- class **boost::serialization::access**
- class **Importer_LIST**
- class **Importer_SET**
- class **Importer_MAP_int**
- std::ostream & operator<< (std::ostream &os, const AgentRequest &request)

  Prints the specified AgentRequest to the specified ostream.

5.9.1 Detailed Description

Encapsulates a request made by one process for agents in another.

Includes a list of requests and a list that represents cancellations of previous requests.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 repast::AgentRequest::AgentRequest (int *sourceProcess*)

Creates an AgentRequest that comes from the specified process.

Parameters

sourceProcess	the rank of the process making the request

5.9.2.2 repast::AgentRequest::AgentRequest (int *sourceProcess,* int *targetProcess*)

Creates an AgentRequest made from the source process to the target process.

This can be used when all the requested agents reside on the same process (i.e. the target process).

Parameters

sourceProcess	the rank of the source process
targetProcess	the rank of the target process

5.9.3 Member Function Documentation

5.9.3.1 void repast::AgentRequest::addAll (const **AgentRequest** & *req*)

Adds all the agent ids (both requests and cancellations) in req to this AgentRequest.

Parameters

req	the AgentRequest to add all the agent ids from

5.9.3.2 void repast::AgentRequest::addAllCancellations (const **AgentRequest** & *req*)

Adds all the agent ids in req to this request, including only the ids that are cancellations and not those that are requests.

Parameters

req	the AgentRequest to add all the agent ids from

5.9.3.3 void repast::AgentRequest::addAllRequests (const **AgentRequest** & *req*)

Adds all the agent ids in req to this request, including only the ids that are requests and not those that are cancellations.

Parameters

req	the AgentRequest to add all the agent ids from

5.9.3.4 void repast::AgentRequest::addCancellation (const **AgentId** & *id*)

Adds the specified agent to the collection of agents for which a previous request is being cancelled.

Parameters

id	the AgentId of the agent for which the request is being cancelled

5.9.3.5 void repast::AgentRequest::addRequest (const **AgentId** & *id*)

Adds the specified agent to the collection agents being requested.

Parameters

id	the requested agent

5.9.3.6 const std::vector<AgentId>& repast::AgentRequest::cancellations ( ) const [inline]

Gets a reference to the vector of cancellations.

Returns

a reference to the vector of AgentIds representing cancellations.

5.9.3.7 bool repast::AgentRequest::contains (const AgentId & *id*)

Returns true if this AgentRequest contains a request for the specified id (either a request or a cancellation), otherwise false.

Parameters

id	the id sought in the lists of requests and cancellations

Returns

true if either the list of requests or the list of cancellations contains the specified id

5.9.3.8 bool repast::AgentRequest::containsInCancellations (const AgentId & *id*)

Returns true if the list of cancellations contains the specified id (the list of requests is ignored)

Parameters

id	the AgentId sought

Returns

true if the specified AgentId is in the list of cancellations

5.9.3.9 bool repast::AgentRequest::containsInRequests (const AgentId & *id*)

Returns true if the list of requests contains the specified id (the list of cancellations is ignored)

Parameters

id	the AgentId sought

Returns

true if the specified AgentId is in the list of requests

5.9.3.10 bool repast::AgentRequest::remove (const AgentId & *id*, bool *removeAllInstances* = true)

Removes the specified id from the lists of requested agents, including both requests and cancellations.

Parameters

id	the AgentId to be removed
removeAll-Instances	if true (the default), all instances of the AgentId are removed; if false, only the first instance found is removed

Returns

true if the id was found (in either list) and removed, otherwise false.

5.9.3.11 bool repast::AgentRequest::removeCancellation (const AgentId & *id*, bool *removeAllInstances* = `true`)

Removes the specified id from the list of agent request cancellations; does not affect the list of requests.

Parameters

id	the AgentId to be removed
removeAll-Instances	if true (the default), all instances of the AgentId are removed; if false, only the first instance found is removed

Returns

true if the id was found in the list of cancellations and removed, otherwise false

5.9.3.12 bool repast::AgentRequest::removeRequest (const AgentId & *id*, bool *removeAllInstances* = `true`)

Removes the specified id from the list of agent requests; does not affect the list of cancellations.

Parameters

id	the AgentId to be removed
removeAll-Instances	if true (the default), all instances of the AgentId are removed; if false, only the first instance found is removed

Returns

true if the id was found in the list of requests and removed, otherwise false

5.9.3.13 int repast::AgentRequest::requestCount ( ) const `[inline]`

Gets the number agents requested.

Includes both requests and cancellations; exactly equivalent to

requestCountRequested() + requestCountCancellations()

Returns

the number agents requested.

5.9.3.14 int repast::AgentRequest::requestCountCancellations ( ) const `[inline]`

Gets the number of agents requested, counting only the cancellations and not the requests.

Returns

the number of agents requested (cancellations only)

5.9.3.15 int repast::AgentRequest::requestCountRequested ( ) const [inline]

Gets the number of agents requested, counting only the requests and not the cancellations.

Returns

the number of agents requested (requests only)

5.9.3.16 const std::vector<AgentId>& repast::AgentRequest::requestedAgents ( ) const [inline]

Gets a reference to the vector of requested agents.

Returns

a reference to the vector of requested agents.

5.9.3.17 int repast::AgentRequest::sourceProcess ( ) const [inline]

Gets the source process of these requests, that is, the process making the request.

Returns

the process making the request

5.9.3.18 int repast::AgentRequest::targetProcess ( ) const [inline]

If the requested agent ids are all on the same process then target process will identify that process.

Otherwise this will return -1.

5.9.3.19 void repast::AgentRequest::targets (std::set< int > & *targets*)

Puts the targets of all the requests into the set.

Includes both the requests and the cancellations.

Parameters

targets	set into which targets will be placed

5.9.3.20 void repast::AgentRequest::targetsOfCancellations (std::set< int > & *targets*)

Puts the targets of all the requests into the set, including only the requests and not the cancellations.

Parameters

targets	the set into which the targets will be placed

5.9.3.21 void repast::AgentRequest::targetsOfRequests (std::set< int > & *targets*)

Puts the targets of all the requests into the set, including only the requests and not the cancellations.

Parameters

targets	the set into which the targets will be placed

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

- repast_hpc/AgentRequest.h
- repast_hpc/AgentRequest.cpp

5.10 repast::AgentStateFilter< T > Struct Template Reference

Used in a filter iterator to filter on local or non-local agents only.

```
#include <SharedContext.h>
```

Public Member Functions

- **AgentStateFilter** (int rankInCommunicator)
- **AgentStateFilter** (bool localFlag, int rankInCommunicator)
- bool **operator()** (const boost::shared_ptr< T > &ptr)

Public Attributes

- int **rank**
- bool **local**

5.10.1 Detailed Description

template<typename T>struct repast::AgentStateFilter< T >

Used in a filter iterator to filter on local or non-local agents only.

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

- repast_hpc/SharedContext.h

5.11 repast::AgentStatus Class Reference

Encapsulates the status (moved or removed) of agent in order to synchronize that status across processes.

```
#include <AgentStatus.h>
```

Public Types

- enum Status { **REMOVED**, **MOVED** }

  Enum indicating the status of th agent.

Public Member Functions

- AgentStatus ()

  No-arg constructor for serialization.
- AgentStatus (AgentId id)

Creates an AgentStatus indicating the status for the specified agent.

- AgentStatus (AgentId old, AgentId newId)

  Creates an AgentStatus indicating the status for the specified agent and the new id of that agent as result from the change in status.
- Status getStatus () const

  Gets the status.
- const AgentId & getId () const

  Gets the id of the agent that this is the status for.
- const AgentId & getOldId () const

  Gets the old id of the agent that this is the status for, if this contains an old and updated AgentId.
- const AgentId & getNewId () const

  Gets the new updated id of the agent that this is the status for, if this contains an old and updated AgentId.

Friends

- class **boost::serialization::access**
- bool operator< (const AgentStatus &one, const AgentStatus &two)

  Comparison operator that can be used in sorts, etc.

5.11.1 Detailed Description

Encapsulates the status (moved or removed) of agent in order to synchronize that status across processes.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 repast::AgentStatus::AgentStatus (AgentId *id*)

Creates an AgentStatus indicating the status for the specified agent.

Parameters

id	the id of the agent whose status this represents

5.11.2.2 repast::AgentStatus::AgentStatus (AgentId *old*, AgentId *newId*)

Creates an AgentStatus indicating the status for the specified agent and the new id of that agent as result from the change in status.

When an agent moves between processes its current rank may change and thus the current rank part of its id will change.

Parameters

old	the id of the agent whose status this represents
newId	the new id of the agent that results from its status change

5.11.3 Member Function Documentation

5.11.3.1 const AgentId& repast::AgentStatus::getId ( ) const [inline]

Gets the id of the agent that this is the status for.

Returns

the id of the agent that this is the status for.

5.11.3.2 const AgentId& repast::AgentStatus::getNewId ( ) const [inline]

Gets the new updated id of the agent that this is the status for, if this contains an old and updated AgentId.

Returns

Gets the new id of the agent that this is the status for, if this contains an old and updated AgentId.

5.11.3.3 const AgentId& repast::AgentStatus::getOldId ( ) const [inline]

Gets the old id of the agent that this is the status for, if this contains an old and updated AgentId.

Returns

Gets the old id of the agent that this is the status for, if this contains an old and updated AgentId.

5.11.3.4 Status repast::AgentStatus::getStatus ( ) const [inline]

Gets the status.

Returns

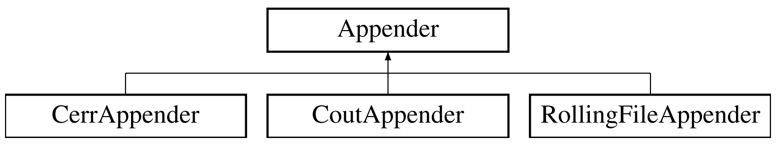
the status

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

- repast_hpc/AgentStatus.h
- repast_hpc/AgentStatus.cpp

5.12 Appender Class Reference

Inheritance diagram for Appender:



Public Member Functions

- **Appender** (const std::string name)
- virtual void **write** (const std::string &line)=0
- virtual void **close** ()
- const std::string & **name** () const

Protected Attributes

- const std::string **_name**

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

- repast_hpc/logger.h
- repast_hpc/logger.cpp

5.13 AppenderBuilder Class Reference

Public Member Functions

- **AppenderBuilder** (const std::string name)
- Appender $*$ **build** ()

Public Attributes

- std::string **name**
- std::string **file_name**
- long **max_size**
- int **max_idx**

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

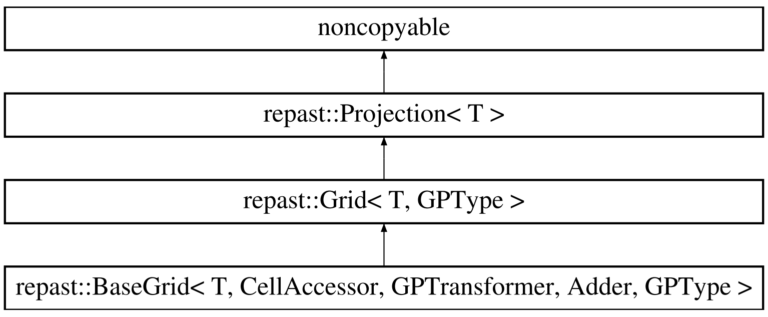
- repast_hpc/logger.h
- repast_hpc/logger.cpp

5.14 repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType > Class Template Reference

Base grid implementation, implementing elements common to both Grids and ContinuousSpaces.

```
#include <BaseGrid.h>
```

Inheritance diagram for repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >:


noncopyable
repast::Projection< T >
repast::Grid< T, GPType >
repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >

Public Types

- typedef
  boost::transform_iterator
  < AgentFromGridPoint< T,
  GPType >, LocationMapConstIter > const_iterator

A const iterator over shared_ptr< T>.

Public Member Functions

- BaseGrid (std::string name, GridDimensions dimensions)

  Creates a BaseGrid with the specified name and dimensions.
- virtual bool contains (const AgentId &id)

  Gets whether or not this grid contains the agent with the specified id.
- virtual bool getLocation (const T ∗agent, std::vector< GPType > &pt) const

  Gets the location of this agent and puts it in the specified vector.
- virtual bool getLocation (const AgentId &id, std::vector< GPType > &out) const

  Gets the location of this agent and puts it in the specified vectors.
- virtual T ∗ getObjectAt (const Point< GPType > &pt) const

  Gets the first object found at the specified point, or NULL if there is no such object.
- virtual void getObjectsAt (const Point< GPType > &pt, std::vector< T ∗ > &out) const

  Gets all the objects found at the specified point.
- virtual bool moveTo (const T ∗agent, const std::vector< GPType > &newLocation)

  Moves the specified agent to the specified location.
- virtual bool moveTo (const T ∗agent, const Point< GPType > &newLocation)

  Moves the specified agent to the specified location.
- virtual bool moveTo (const AgentId &id, const std::vector< GPType > &newLocation)

  Moves the specified agent to the specified location.
- virtual bool moveTo (const AgentId &id, const Point< GPType > &pt)

  Moves the specified agent to the specified point.
- virtual std::pair< bool, Point
  < GPType > > moveByDisplacement (const T ∗agent, const std::vector< GPType > &displacement)

  Moves the specified object from its current location by the specified amount.
- virtual std::pair< bool, Point
  < GPType > > moveByVector (const T ∗agent, double distance, const std::vector< double > &anglesIn-Radians)

  doc inherited from Grid
- virtual const_iterator begin () const

  Gets an iterator over the agents in this BaseGrid starting with the first agent.
- virtual const_iterator end () const

  Gets the end of an iterator over the agents in this BaseGrid.
- virtual size_t size () const

  Gets the number of agents in this BaseGrid.
- virtual double getDistance (const Point< GPType > &pt1, const Point< GPType > &pt2) const

  Gets the distance between the two grid points.
- virtual double getDistanceSq (const Point< GPType > &pt1, const Point< GPType > &pt2) const

  Gets the square of the distance between the two grid points.
- virtual void getDisplacement (const Point< GPType > &pt1, const Point< GPType > &pt2, std::vector< GPType > &out) const

  Gets vector difference between point 1 and point 2, putting the result in out.
- virtual const GridDimensions dimensions () const

  Gets the dimensions of this Grid.
- virtual void translate (const Point< GPType > &location, const Point< GPType > &displacement, std-::vector< GPType > &out) const

  Translates the specified location by the specified displacement put the result in out.
- virtual void transform (const std::vector< GPType > &location, std::vector< GPType > &out) const

  Transforms the specified location using the properties (e.g.

- virtual bool isPeriodic () const

  Gets whether or not this grid is periodic (i.e.
- virtual ProjectionInfoPacket ∗ **getProjectionInfo** (AgentId id, bool secondaryInfo=false, std::set< AgentId > ∗secondaryIds=0, int destProc=-1)
- virtual void **updateProjectionInfo** (ProjectionInfoPacket ∗pip, Context< T > ∗context)
- virtual void getAgentsToPush (std::set< AgentId > &agentsToTest, std::map< int, std::set< AgentId > > &agentsToPush)

  Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.

Protected Types

- typedef AgentLocationMap::iterator **LocationMapIter**
- typedef
  AgentLocationMap::const_iterator **LocationMapConstIter**

Protected Member Functions

- virtual bool **addAgent** (boost::shared_ptr< T > agent)
- virtual void **removeAgent** (T ∗agent)
- LocationMapConstIter **locationsBegin** () const
- LocationMapConstIter **locationsEnd** () const
- T ∗ **get** (const AgentId &id)

Protected Attributes

- GPTransformer **gpTransformer**
- Adder **adder**

5.14.1 Detailed Description

template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType>class repast::-BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >

Base grid implementation, implementing elements common to both Grids and ContinuousSpaces.

Standard grid and space types that provide defaults for the various template parameters can be found in Space in Space.h

Template Parameters

T	the type of objects contained by this BaseGrid (generally the type of agents)
CellAccessor	implements the actual storage for the grid.
GPTransformer	transforms cell points according to the topology (e.g. periodic) of the BaseGrid.
Adder	determines how objects are added to the grid from its associated context.
GPType	the coordinate type of the grid point locations; this must be an int or a double.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 template<typename T , typename CellAccessor , typename GPTransformer , typename Adder , typename GPType > repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::BaseGrid (std::string *name*, GridDimensions *dimensions*)

Creates a BaseGrid with the specified name and dimensions.

Parameters

name	the name of the BaseGrid
dimensions	the dimensions of the BaseGrid

5.14.3 Member Function Documentation

5.14.3.1 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual const_iterator repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::begin ( ) const `[inline]`, `[virtual]`

Gets an iterator over the agents in this BaseGrid starting with the first agent.

The iterator derefrences into shared_ptr<T>. The actual agent can be accessed by derefrenceing the iter: (∗iter)->getId() for example.

Returns

an iterator over the agents in this BaseGrid starting with the first agent.

5.14.3.2 template<typename T , typename CellAccessor , typename GPTransformer , typename Adder , typename GPType > bool repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::contains (const AgentId & *id*) `[virtual]`

Gets whether or not this grid contains the agent with the specified id.

Parameters

id	the id of the agent to check

Returns

true if the grid contains the agent, otherwise false.

Implements repast::Grid< T, GPType >.

5.14.3.3 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual const GridDimensions repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::dimensions ( ) const `[inline]`, `[virtual]`

Gets the dimensions of this Grid.

Returns

the dimensions of this Grid.

Implements repast::Grid< T, GPType >.

Reimplemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, and repast::SharedBaseGrid< T, GPTransformer, Adder, double >.

5.14.3.4 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual const_iterator repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::end ( ) const `[inline]`, `[virtual]`

Gets the end of an iterator over the agents in this BaseGrid.

Returns

the end of an iterator over the agents in this BaseGrid.

5.14.3.5 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual void repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::getAgentsToPush (std::set< AgentId > & *agentsToTest,* std::map< int, std::set< AgentId > > & *agentsToPush*) `[inline]`, `[virtual]`

Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.

Generally spaces must push agents that are in 'buffer zones' and graphs must push local agents that are vertices to master edges where the other vertex is non- local. The results are returned per-process in the agentsToPush map.

Implements repast::Grid< T, GPType >.

Reimplemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, repast::SharedBaseGrid< T, GPTransformer, Adder, double >, and repast::SharedDiscreteSpace< T, GPTransformer, Adder >.

5.14.3.6 template<typename T , typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> void repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::getDisplacement (const Point< GPType > & *pt1,* const Point< GPType > & *pt2,* std::vector< GPType > & *out*) const `[virtual]`

Gets vector difference between point 1 and point 2, putting the result in out.

Parameters

	p1	the first point
	p2	the second point
`out`	*the*	vector where the difference will be put

Implements repast::Grid< T, GPType >.

5.14.3.7 template<typename T , typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> double repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::getDistance (const Point< GPType > & *pt1,* const Point< GPType > & *pt2*) const `[virtual]`

Gets the distance between the two grid points.

Parameters

p1	the first point
p2	the second point

Returns

the distance between pt1 and pt2.

Implements repast::Grid< T, GPType >.

5.14.3.8 template<typename T , typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> double repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::getDistanceSq (const Point< GPType > & *pt1,* const Point< GPType > & *pt2*) const `[virtual]`

Gets the square of the distance between the two grid points.

Parameters

p1	the first point
p2	the second point

Returns

the square of the distance between pt1 and pt2.

Implements repast::Grid< T, GPType >.

5.14.3.9 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> bool **repast::BaseGrid**< T, CellAccessor, GPTransformer, Adder, GPType >::getLocation (const T ∗ *agent*, std::vector< GPType > & *out*) const `[virtual]`

Gets the location of this agent and puts it in the specified vector.

The x coordinate will be the first value, the y the second and so on.

Parameters

	agent	the agent whose location we want to get
`out`	*the*	vector where the agents location will be put

Returns

true if the location was successfully found, otherwise false.

Implements repast::Grid< T, GPType >.

5.14.3.10 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> bool **repast::BaseGrid**< T, CellAccessor, GPTransformer, Adder, GPType >::getLocation (const **AgentId** & *id*, std::vector< GPType > & *out*) const `[virtual]`

Gets the location of this agent and puts it in the specified vectors.

The x coordinate will be the first value, the y the second and so on.

Parameters

	id	the id of the agent whose location we want to get
`out`	*out*	the agent's location will be put into this vector

Returns

true if the location was successfully found, otherwise false.

Implements repast::Grid< T, GPType >.

5.14.3.11 template<typename T , typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> T ∗ **repast::BaseGrid**< T, CellAccessor, GPTransformer, Adder, GPType >::getObjectAt (const **Point**< GPType > & *pt*) const `[virtual]`

Gets the first object found at the specified point, or NULL if there is no such object.

Returns

the first object found at the specified point, or NULL if there is no such object.

Implements repast::Grid< T, GPType >.

5.14.3.12 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> void **repast::BaseGrid**< T, CellAccessor, GPTransformer, Adder, GPType >::getObjectsAt (const **Point**< GPType > & *pt*, std::vector< T ∗ > & *out*) const `[virtual]`

Gets all the objects found at the specified point.

The found objects will be put into the out parameter.

Parameters

	pt	the point to get all the objects at
`out`	*out*	the vector into which the found objects will be put

Implements repast::Grid< T, GPType >.

5.14.3.13 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual bool **repast::BaseGrid**< T, CellAccessor, GPTransformer, Adder, GPType >::isPeriodic ( ) const `[inline]`, `[virtual]`

Gets whether or not this grid is periodic (i.e.

toroidal).

Returns

true if this Grid is periodic, otherwise false.

Implements repast::Grid< T, GPType >.

5.14.3.14 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> std::pair< bool, **Point**< GPType > > **repast::BaseGrid**< T, CellAccessor, GPTransformer, Adder, GPType >::moveByDisplacement (const T ∗ *agent*, const std::vector< GPType > & *displacement*) `[virtual]`

Moves the specified object from its current location by the specified amount.

For example `moveByDisplacement(object, 3, -2, 1)` will move the object by 3 along the x-axis, -2 along the y and 1 along the z. The displacement argument can be less than the number of dimensions in the space in which case the remaining argument will be set to 0. For example, `moveByDisplacement(object, 3)` will move the object 3 along the x-axis and 0 along the y and z axes, assuming a 3D grid.

Parameters

agent	the object to move
displacement	the amount to move the object

Returns

a pair containing a bool that indicates whether the move was a success or not, and the point where the agent was moved to.

Implements repast::Grid< T, GPType >.

5.14.3.15 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> bool **repast::BaseGrid**< T, CellAccessor, GPTransformer, Adder, GPType >::moveTo (const T ∗ *agent*, const std::vector< GPType > & *newLocation*) `[virtual]`

Moves the specified agent to the specified location.

Returns true if the move was successful otherwise false. The agent must be already added to the context associated with this space, otherwise this throws an out_of_range exception if the new location out of bounds.

Parameters

agent	the agent to move
newLocation	the location to move to

Returns

true if the move was successful, otherwise false

5.14.3.16 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> bool repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::moveTo (const T ∗ *agent,* const Point< GPType > & *newLocation*) `[virtual]`

Moves the specified agent to the specified location.

Returns true if the move was successful otherwise false. The agent must be already added to the context associated with this space, otherwise this throws an out_of_range exception if the new location out of bounds.

Parameters

agent	the agent to move
newLocation	the location to move to

Returns

true if the move was successful, otherwise false

5.14.3.17 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> bool repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::moveTo (const AgentId & *id,* const std::vector< GPType > & *newLocation*) `[virtual]`

Moves the specified agent to the specified location.

Returns true if the move was successful otherwise false. The agent must be already added to the context associated with this space, otherwise this throws an out_of_range exception if the new location out of bounds.

Parameters

id	the id of the agent to move
newLocation	the location to move to

Returns

true if the move was successful, otherwise false

Reimplemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, and repast::SharedBaseGrid< T, GPTransformer, Adder, double >.

5.14.3.18 template<typename T, typename CellAccessor , typename GPTransformer , typename Adder , typename GPType> bool repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::moveTo (const AgentId & *id,* const Point< GPType > & *pt*) `[virtual]`

Moves the specified agent to the specified point.

Parameters

id	the id of the agent to move
pt	where to move the agent to

Returns

true if the move was successful, otherwise false

Implements repast::Grid< T, GPType >.

Reimplemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, and repast::SharedBaseGrid< T, GPTransformer, Adder, double >.

5.14.3.19 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual size_t repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::size ( ) const `[inline]`, `[virtual]`

Gets the number of agents in this BaseGrid.

Returns

the number of agents in this BaseGrid.

5.14.3.20 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual void repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::transform (const std::vector< GPType > & *location*, std::vector< GPType > & *out*) const `[inline]`, `[virtual]`

Transforms the specified location using the properties (e.g.

toroidal) of this space.

Parameters

	location	the location to transform
`out`	*out*	the vector where the result of the transform will be put

Implements repast::Grid< T, GPType >.

5.14.3.21 template<typename T, typename CellAccessor, typename GPTransformer, typename Adder, typename GPType> virtual void repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >::translate (const Point< GPType > & *location*, const Point< GPType > & *displacement*, std::vector< GPType > & *out*) const `[inline]`, `[virtual]`

Translates the specified location by the specified displacement put the result in out.

Parameters

	location	the initial location
	displacement	the amount to translate the location by
`out`	*out*	the vector where the result of the translation is put

Implements repast::Grid< T, GPType >.

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

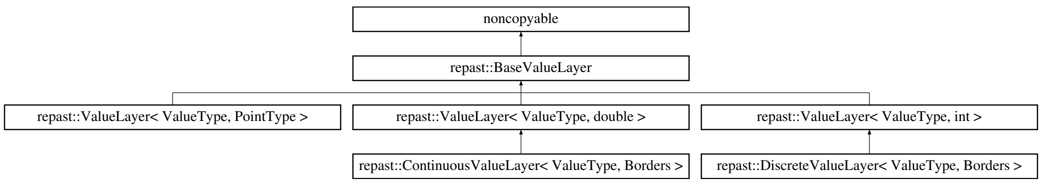
- repast_hpc/BaseGrid.h

5.15 repast::BaseValueLayer Class Reference

Base implementation of a ValueLayer.

`#include <ValueLayer.h>`

Inheritance diagram for repast::BaseValueLayer:


noncopyable
repast::BaseValueLayer
repast::ValueLayer< ValueType, PointType >
repast::ValueLayer< ValueType, double >
repast::ValueLayer< ValueType, int >
repast::ContinuousValueLayer< ValueType, Borders >
repast::DiscreteValueLayer< ValueType, Borders >

Public Member Functions

- BaseValueLayer (const std::string &name)

    Creates a BaseValueLayer with the specified name.
- std::string name () const

    Gets the value layer's name.

Protected Attributes

- std::string **_name**

5.15.1 Detailed Description

Base implementation of a ValueLayer.

A ValueLayer stores values by location.

5.15.2 Member Function Documentation

5.15.2.1 std::string repast::BaseValueLayer::name ( ) const [inline]

Gets the value layer's name.

Returns

the name of the value layer.

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

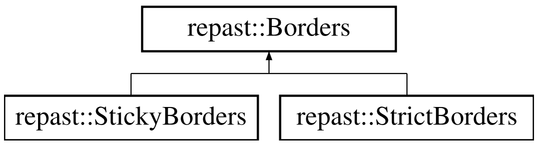
- repast_hpc/ValueLayer.h
- repast_hpc/ValueLayer.cpp

5.16 repast::Borders Class Reference

Base class for representations of border semantics (e.g.

`#include <GridComponents.h>`

Inheritance diagram for repast::Borders:



Public Member Functions

- **Borders** (GridDimensions d)
- void **transform** (const std::vector< int > &in, std::vector< int > &out) const
- void **transform** (const std::vector< double > &in, std::vector< double > &out) const
- bool **isPeriodic** () const

Protected Member Functions

- void **boundsCheck** (const std::vector< int > &pt) const
- void **boundsCheck** (const std::vector< double > &pt) const

Protected Attributes

- const GridDimensions **_dimensions**

5.16.1 Detailed Description

Base class for representations of border semantics (e.g.

Strict, Sticky, etc.)

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

- repast_hpc/GridComponents.h
- repast_hpc/GridComponents.cpp

5.17 repast::CartTopology Class Reference

Allows retrieval of the position of this process within the MPI Cartesian Topology into which it is placed.

```
#include <SharedBaseGrid.h>
```

Public Member Functions

- **CartTopology** (std::vector< int > processesPerDim, std::vector< double > origin, std::vector< double > extents, bool spaceIsPeriodic, boost::mpi::communicator ∗world)
- void getCoordinates (int rank, std::vector< int > &coords)

  Gets the coordinates in the MPI Cartesian Communicator for the specified rank.
- GridDimensions getDimensions (int rank)

  Gets the GridDimensions boundaries for the specified rank.
- GridDimensions getDimensions (std::vector< int > &pCoordinates)

  Gets the GridDimensions boundaries for the specified MPI coordinates.
- void **createNeighbors** (Neighbors &nghs)

5.17.1 Detailed Description

Allows retrieval of the position of this process within the MPI Cartesian Topology into which it is placed.

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

- repast_hpc/SharedBaseGrid.h
- repast_hpc/SharedBaseGrid.cpp

5.18 repast::CellContents< AgentContent, GPType > Class Template Reference

DEPRECATED Encapsulates the contents of a grid / space location so that it can be sent between processes.

`#include <SharedBaseGrid.h>`

Public Member Functions

- template<class Archive >
  void **serialize** (Archive &ar, const unsigned int version)
- **CellContents** (Point< GPType > pt)

Public Attributes

- Point< GPType > **_pt**
- std::vector< AgentContent > **_objs**

Friends

- class **boost::serialization::access**

5.18.1 Detailed Description

template<typename AgentContent, typename GPType>class repast::CellContents< AgentContent, GPType >

DEPRECATED Encapsulates the contents of a grid / space location so that it can be sent between processes.

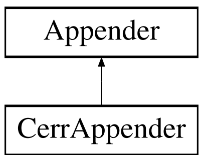
Deprecated Replaced by ProjectionInfoPacket as of Version 2.0

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

- repast_hpc/SharedBaseGrid.h

5.19 CerrAppender Class Reference

Inheritance diagram for CerrAppender:



Public Member Functions

- void **write** (const string &line)

Additional Inherited Members

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

- repast_hpc/logger.cpp

5.20 ConfigLexer Class Reference

Public Member Functions

- **ConfigLexer** (const string &file_name, boost::mpi::communicator ∗comm=0, int maxConfigFileSize=MAX_-CONFIG_FILE_SIZE)
- TOKEN **next_token** ()
- string **key** ()
- string **value** ()
- string **error** ()
- int **line** ()
- void **reset** ()

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

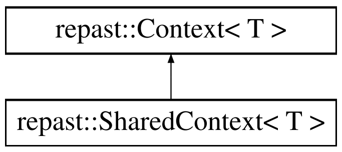
- repast_hpc/logger.cpp

5.21 repast::Context< T > Class Template Reference

Collection of agents of type T with set semantics.

`#include <Context.h>`

Inheritance diagram for repast::Context< T >:



Public Types

- typedef boost::transform_iterator < SecondElement< T >, typename AgentMap::const_iterator > **const_iterator**
- typedef boost::filter_iterator < IsAgentType< T >, typename Context< T >::const_iterator > **const_bytype_iterator**

Public Member Functions

- virtual ∼Context ()

  Destroys this context and the projections it contains.
- T ∗ addAgent (T ∗agent)

  Adds the agent to the context.
- virtual void addProjection (Projection< T > ∗projection)

  Adds the specified projection to this context.
- Projection< T > ∗ getProjection (const std::string &name)

  Get the named Projection.
- void removeAgent (const AgentId id)

  Removes the specified agent from this context.
- void removeAgent (T ∗agent)

  Removes the specified agent from this context.
- T ∗ getAgent (const AgentId &id)

  Gets the specified agent.
- void getRandomAgents (const int count, std::vector< T ∗ > &agents)

  Gets at random the specified count of agents and returns them in the agents vector.
- const_iterator begin () const

  Gets the start of iterator over the agents in this context.
- const_iterator end () const

  Gets the end of an iterator over the agents in this context.
- const_bytype_iterator byTypeBegin (int typeId) const

  Gets the start of an iterator over agents in this context of the specified type.
- const_bytype_iterator byTypeEnd (int typeId) const

  Gets the end of an iterator over agents in this context of the specified type.
- bool contains (const AgentId &id)

  Returns true if the specified agent is in this context, otherwise false.
- int size () const

  Gets the size (number of agents) in this context.
- void addValueLayer (BaseValueLayer ∗valueLayer)

  Adds a value layer to this context.
- template<typename ValueType , typename Borders >
  DiscreteValueLayer< ValueType,
  Borders > ∗ getDiscreteValueLayer (const std::string &valueLayerName)

  Gets the named discrete value layer from this Context.
- template<typename ValueType , typename Borders >
  ContinuousValueLayer
  < ValueType, Borders > ∗ getContinuousValueLayer (const std::string &valueLayerName)

  Gets the named continuous value layer from this Context.
- template<typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  Context< T >::const_iterator > filteredBegin (const filterStruct &fStruct)

  Creates a filtered iterator over the set of agents in this context and returns it with a value equal to the beginning of the list.
- template<typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  Context< T >::const_iterator > filteredEnd (const filterStruct &fStruct)

  Creates a filtered iterator over the set of agents in this context and returns it with a value equal to one step past end of the list.

- template<typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  Context< T >
  ::const_bytype_iterator > byTypeFilteredBegin (const int type, const filterStruct &fStruct)

  Creates a filtered iterator over the set of agents in this context of the specified type (per their AgentId values), and returns it with a value equal to the beginning of the list.
- template<typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  Context< T >
  ::const_bytype_iterator > byTypeFilteredEnd (const int type, const filterStruct &fStruct)

  Creates a filtered iterator over the set of agents in this context of the specified type (per their AgentId values), and returns it with a value equal to one past the end of the list.
- void selectAgents (std::set< T ∗ > &selectedAgents, bool remove=false)

  Gets a set of pointers to all agents in this context.
- void selectAgents (std::vector< T ∗ > &selectedAgents, bool remove=false)

  Gets a randomly ordered vector of pointers to all agents in this context.
- void selectAgents (int count, std::set< T ∗ > &selectedAgents, bool remove=false)

  Gets a set of pointers to a specified number of randomly selected agents.
- void selectAgents (int count, std::vector< T ∗ > &selectedAgents, bool remove=false)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected agents.
- void selectAgents (std::set< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a set of pointers to all agents in this context of a specified type (per their AgentId values).
- void selectAgents (std::vector< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to all agents in this context of a specified type (per their AgentId values).
- void selectAgents (int count, std::set< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a set of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values).
- void selectAgents (int count, std::vector< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values).
- template<typename filterStruct >
  void selectAgents (std::set< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to all agents in this context matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (std::vector< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to all agents in this context matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (int count, std::set< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to a specified number of randomly selected agents matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (int count, std::vector< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected agents matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (std::set< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to all agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (std::vector< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

Gets a randomly ordered vector of pointers to all agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.

- template<typename filterStruct >
  void selectAgents (int count, std::set< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values) and matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (int count, std::vector< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values) and matching a user-defined filter.
- void getProjectionInfo (AgentRequest req, std::map< std::string, std::vector< repast::ProjectionInfoPacket ∗ > > &map, bool secondaryInfo=false, std::set< AgentId > ∗secondaryIds=0, int destProc=-1)

  Gets the projection information for all projections in this context, for all agents whose IDs are listed in the AgentRequest.
- void setProjectionInfo (std::map< std::string, std::vector< repast::ProjectionInfoPacket ∗ > > &projInfo)

  Sets the projection information as specified.
- void **cleanProjectionInfo** (std::set< AgentId > &agentsToKeep)

Protected Attributes

- std::vector< Projection< T > ∗ > **projections**

5.21.1 Detailed Description

template<typename T>class repast::Context< T >

Collection of agents of type T with set semantics.

Object identity and equality is determined by their AgentId.

Template Parameters

the	type objects contained by the Context. The T must extends repast::Agent.

5.21.2 Member Function Documentation

5.21.2.1 template<typename T > T ∗ repast::Context< T >::addAgent (T ∗ *agent*)

Adds the agent to the context.

Performs a check to ensure that no agent with the same ID (presumably the 'same' agent) has previously been added. If a matching ID is found, the new agent is not added, and the address of the pre-existing agent is returned. If no match is found, the agent is added and the return value is the same as the value passed

Parameters

agent	the agent to add

Returns

the address of the agent in the context; will be the same as the address passed if the agent was successfully added, but if there was already an agent with the same ID the address returned will be that of the pre-existing agent, which is not replaced.

5.21.2.2 template<typename T > void repast::Context< T >::addProjection (Projection< T > ∗ *projection*) [virtual]

Adds the specified projection to this context.

All the agents in this context will be added to the Projection. Any agents subsequently added to this context will also be added to the Projection.

Parameters

projection	the projection to add

Reimplemented in repast::SharedContext< T >.

5.21.2.3 template<typename T > void repast::Context< T >::addValueLayer (BaseValueLayer ∗ *valueLayer*)

Adds a value layer to this context.

Parameters

valueLayer	the value layer to add

5.21.2.4 template<typename T> const_iterator repast::Context< T >::begin ( ) const [inline]

Gets the start of iterator over the agents in this context.

The iterator derefrences into shared_ptr<T>. The actual agent can be accessed by derefrenceing the iter: (∗iter)->getId() for example.

Returns

the start of iterator over the agents in this context.

5.21.2.5 template<typename T> const_bytype_iterator repast::Context< T >::byTypeBegin (int *typeId*) const [inline]

Gets the start of an iterator over agents in this context of the specified type.

The type corresponds to the type component of an agent's AgentId.

Parameters

typeId	the type of the agent. Only Agents whose agentId.agentType() is equal to this typeId will be included in the iterator

Returns

the start of an iterator over agents in this context of the specified type.

5.21.2.6 template<typename T> const_bytype_iterator repast::Context< T >::byTypeEnd (int *typeId*) const [inline]

Gets the end of an iterator over agents in this context of the specified type.

The type corresponds to the type component of an agent's AgentId.

Parameters

typeId	the type of the agent. Only Agents whose agentId.agentType() is equal to this typeId will be included in the iterator

Returns

the end of an iterator over agents in this context of the specified type.

5.21.2.7 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename **Context**< T >::const_bytype_iterator > **repast::Context**< T >::byTypeFilteredBegin (const int *type,* const filterStruct & *fStruct*)

Creates a filtered iterator over the set of agents in this context of the specified type (per their AgentId values), and returns it with a value equal to the beginning of the list.

The struct can be any user-defined structure that implements a unary operator (see IsAgentType) that can be passed and which will become a filter to sort across the agent list, e.g.:

struct filter { bool operator()(const boost::shared_ptr<T>& ptr){ return (ptr->getAgentValue() == targetValue;) } }

This should allow filtering of agents by type and on any attribute.

Parameters

fStruct	an instance of the struct to be used as the filter
type	the numeric type of agents to be included in the list

Template Parameters

filterStruct	the type of the filter to be applied to the agents

Returns

an iterator positioned at the beginning of the list of agents meeting the filter's criteria

5.21.2.8 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename **Context**< T >::const_bytype_iterator > **repast::Context**< T >::byTypeFilteredEnd (const int *type,* const filterStruct & *fStruct*)

Creates a filtered iterator over the set of agents in this context of the specified type (per their AgentId values), and returns it with a value equal to one past the end of the list.

The struct can be any user-defined structure that implements a unary operator (see IsAgentType) that can be passed and which will become a filter to sort across the agent list, e.g.:

struct filter { bool operator()(const boost::shared_ptr<T>& ptr){ return (ptr->getAgentValue() == targetValue;) } }

This should allow filtering of agents by type and on any attribute.

Parameters

fStruct	an instance of the struct to be used as the filter
type	the numeric type of agents to be included in the list

Template Parameters

filterStruct	the type of the filter to be applied to the agents

Returns

an iterator positioned at one past the end of the list of agents meeting the filter's criteria

5.21.2.9 template<typename T> const_iterator repast::Context< T >::end ( ) const [inline]

Gets the end of an iterator over the agents in this context.

The iterator derefrences into shared_ptr<T>. The actual agent can be accessed by derefrenceing the iter: (∗iter)->getId() for example.

Returns

the end of an iterator over the agents in this context

5.21.2.10 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename Context< T >::const_iterator > repast::Context< T >::filteredBegin (const filterStruct & *fStruct*)

Creates a filtered iterator over the set of agents in this context and returns it with a value equal to the beginning of the list.

The struct can be any user-defined structure that implements a unary operator (see IsAgentType) that can be passed and which will become a filter to sort across the agent list, e.g.:

struct filter { bool operator()(const boost::shared_ptr<T>& ptr){ return (ptr->getAgentValue() == targetValue;) } }

This should allow filtering of agents by any attribute.

Parameters

fStruct	an instance of the struct to be used as the filter

Template Parameters

filterStruct	the type of the filter to be applied to the agents

Returns

an iterator positioned at the beginning of the list of agents meeting the filter's criteria

5.21.2.11 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename Context< T >::const_iterator > repast::Context< T >::filteredEnd (const filterStruct & *fStruct*)

Creates a filtered iterator over the set of agents in this context and returns it with a value equal to one step past end of the list.

The struct can be any user-defined structure that implements a unary operator (see IsAgentType) that can be passed and which will become a filter to sort across the agent list, e.g.:

struct filter { bool operator()(const boost::shared_ptr<T>& ptr){ return (ptr->getAgentValue() == targetValue;) } }

This should allow filtering of agents by any attribute.

Parameters

fStruct	an instance of the struct to be used as the filter

Template Parameters

filterStruct	the type of the filter to be applied to the agents

Returns

an iterator positioned at one past the end of the list of agents meeting the filter's criteria

5.21.2.12 template< typename T > T * **repast::Context**< T >::getAgent (const **AgentId** & *id*)

Gets the specified agent.

Parameters

the	id of the agent to get.

5.21.2.13 template< typename T > template< typename ValueType , typename Borders > **ContinuousValueLayer**< ValueType, **Borders** > * **repast::Context**< T >::getContinuousValueLayer (const std::string & *valueLayerName*)

Gets the named continuous value layer from this Context.

The value layer must have been previously added.

Parameters

valueLayerName	the name of the value layer to get

Template Parameters

ValueType	the numeric type contained by the value layer
Borders	the Border type of the value layer

Returns

the named continuous value layer from this Context.

5.21.2.14 template< typename T > template< typename ValueType , typename Borders > **DiscreteValueLayer**< ValueType, **Borders** > * **repast::Context**< T >::getDiscreteValueLayer (const std::string & *valueLayerName*)

Gets the named discrete value layer from this Context.

The value layer must have been previously added.

Parameters

valueLayerName	the name of the value layer to get

Template Parameters

ValueType	the numeric type contained by the value layer
Borders	the Border type of the value layer

Returns

the named discrete value layer from this Context.

5.21.2.15 template<typename T > **Projection**< T > ∗ **repast::Context**< T >::getProjection (const std::string & *name*)

Get the named Projection.

Parameters

the	name of the projection to get

Returns

the named Projection or 0 if no such Projection is found.

5.21.2.16 template<typename T > void **repast::Context**< T >::getProjectionInfo (**AgentRequest** *req*, std::map< std::string, std::vector< **repast::ProjectionInfoPacket** ∗ > > & *map*, bool *secondaryInfo* = `false`, std::set< **AgentId** > ∗ *secondaryIds* = `0`, int *destProc* = `-1`)

Gets the projection information for all projections in this context, for all agents whose IDs are listed in the Agent-Request.

The general sense of this method can be easily understood: given a list of agents, get the projection information for all of those agents. But there are some subtleties that should be kept in mind.

"The projection information for an agent" is misleading. In fact, the projection information that is needed can vary depending on the context and on the kind of projection.

Generally speaking, spaces return only one kind projection information: coordinate locations for the agent specified. This is the simplest case.

The more complicated case is given by graphs. A graph projection can return different sets of information depending on how that information will have to be used. The basic issue is that a graph projection returns sets of edges, and edges must be connected to other agents; this means that a mechanism must be in place for ensuring that the projection info that arrives can be used, which means that for a given 'ego' agent, all 'alter' agents that are connected to it by edges must also be on the receiving process. (Note: Repast HPC 1.0 versions sent all of the alter agents' content along with the edge send; this version does not do this, partly to minimize the amount of information being packaged and sent but also because the alternative method used is integrated with the normal bookkeeping for sharing agent information across processes (AgentRequests).) In different circumstances, different assumptions can be made about what information will be available on the receiving process. Note that the coordinate information is generally referred to as 'Primary' information, while edge information is 'secondary'; in a third category ('secondary IDs') are the IDs of the alter agents, which can be packaged separately.

The impact of this is that this function is generally called in the following ways:

1) When requesting agents: in this case, a copy of the agent will be sent from one process to another. No secondary information will be sent at all. This is because it is assumed that if an agent participated in a graph on the receiving process, it would already be present on that process and would not be being requested.

2) When synchronizing Projection Information: in this case, some secondary information (edges) is needed: the edges that connect the specified ego agent with edges on the receiving process. No secondary IDs are needed, because the only edges being sent are those that connect to agents on the receiving process, which will be assumed to already be available on that process.

3) When synchronizing Agent Status (moving agents from process to process): in this case, the full collection of projection information is needed, including all of the edges in which the specified agent participates and all of the secondary IDs. (The secondary IDs of agents that are already on the receiving process can be omitted, at least theoretically.) This allows the full reconstruction of Projection Information on the receiving process.

Parameters

req	List of IDs for agents whose information is requested
map	A map into which the projection information will be placed. Key values represent the names of the projections in this context.
secondaryInfo	true if the 'secondary' projection info must also be returned
secondaryIds	A set of IDs for agents who are referred to by the projection informaton being returned (may be null)
destProc	The Process that will be receiving this information (the information sent may be customized depending on the destination process). If not specified a larger set of information will be sent.

5.21.2.17 template< typename T > void repast::Context< T >::getRandomAgents (const int *count,* std::vector< T * > & *agents*)

Gets at random the specified count of agents and returns them in the agents vector.

Parameters

	count	the number of agents to get
`out`	*agents*	a vector where the agents will be returned

5.21.2.18 template< typename T > void repast::Context< T >::removeAgent (const AgentId *id*)

Removes the specified agent from this context.

Parameters

id	the id of the agent to remove

5.21.2.19 template< typename T > void repast::Context< T >::selectAgents (std::set< T * > & *selectedAgents,* bool *remove* = `false`)

Gets a set of pointers to all agents in this context.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

Parameters

`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)

5.21.2.20 template< typename T > void repast::Context< T >::selectAgents (std::vector< T * > & *selectedAgents,* bool *remove* = `false`)

Gets a randomly ordered vector of pointers to all agents in this context.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

Parameters

`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)

5.21.2.21 template<typename T > void repast::Context< T >::selectAgents (int *count,* std::set< T ∗ > & *selectedAgents,* bool *remove* = `false`)

Gets a set of pointers to a specified number of randomly selected agents.

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)

5.21.2.22 template<typename T > void repast::Context< T >::selectAgents (int *count,* std::vector< T ∗ > & *selectedAgents,* bool *remove* = `false`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected agents.

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)

5.21.2.23 template<typename T > void repast::Context< T >::selectAgents (std::set< T ∗ > & *selectedAgents,* int *type,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to all agents in this context of a specified type (per their AgentId values).

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.21.2.24 template<typename T > void repast::Context< T >::selectAgents (std::vector< T ∗ > & *selectedAgents,* int *type,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to all agents in this context of a specified type (per their AgentId values).

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.21.2.25 template<typename T > void repast::Context< T >::selectAgents (int *count*, std::set< T * > & *selectedAgents*, int *type*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values).

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.21.2.26 template<typename T > void repast::Context< T >::selectAgents (int *count*, std::vector< T * > & *selectedAgents*, int *type*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values).

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.21.2.27 template<typename T > template<typename filterStruct > void repast::Context< T >::selectAgents (std::set< T ∗ > & *selectedAgents,* filterStruct & *filter,* bool *remove =* `false`*,* int *popSize =* `-1`)

Gets a set of pointers to all agents in this context matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

that can possibly be selected, all possible agents will be selected

Parameters

`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.28 template<typename T > template<typename filterStruct > void repast::Context< T >::selectAgents (std::vector< T ∗ > & *selectedAgents,* filterStruct & *filter,* bool *remove =* `false`*,* int *popSize =* `-1`)

Gets a randomly ordered vector of pointers to all agents in this context matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.29 template< typename T > template< typename filterStruct > void repast::Context< T >::selectAgents (int *count*, std::set< T * > & *selectedAgents*, filterStruct & *filter*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to a specified number of randomly selected agents matching a user-defined filter.

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.30 template< typename T > template< typename filterStruct > void repast::Context< T >::selectAgents (int *count*, std::vector< T * > & *selectedAgents*, filterStruct & *filter*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected agents matching a user-defined filter.

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.31 template<typename T > template<typename filterStruct > void repast::Context< T >::selectAgents (std::set< T ∗ > & *selectedAgents*, int *type*, filterStruct & *filter*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to all agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.32 template<typename T > template<typename filterStruct > void repast::Context< T >::selectAgents (std::vector< T ∗ > & *selectedAgents*, int *type*, filterStruct & *filter*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to all agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.33 template<typename T > template<typename filterStruct > void repast::Context< T >::selectAgents (int *count*, std::set< T ∗ > & *selectedAgents*, int *type*, filterStruct & *filter*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values) and matching a user-defined filter.

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.34 template< typename T > template< typename filterStruct > void **repast::Context**< T >::selectAgents (int *count,* std::vector< T * > & *selectedAgents,* int *type,* filterStruct & *filter,* bool *remove* = `false`*,* int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected agents of a specified type (per their AgentId values) and matching a user-defined filter.

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.21.2.35 template< typename T > void **repast::Context**< T >::setProjectionInfo (std::map< std::string, std::vector< **repast::ProjectionInfoPacket** * > > & *projInfo*)

Sets the projection information as specified.

Parameters

projInfo	map where keys represent projections in this context and the values represent collections of projection information content that will be used to specify the relationships among the agents.

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

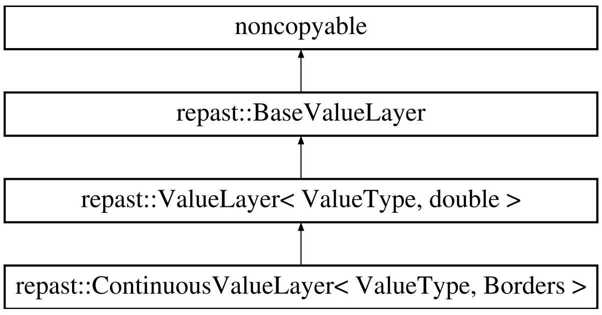
- repast_hpc/Context.h

5.22 repast::ContinuousValueLayer< ValueType, Borders > Class Template Reference

Continous value layer whose location coordinates are double.

`#include <ValueLayer.h>`

Inheritance diagram for repast::ContinuousValueLayer< ValueType, Borders >:


noncopyable
repast::BaseValueLayer
repast::ValueLayer< ValueType, double >
repast::ContinuousValueLayer< ValueType, Borders >

Public Member Functions

- **ContinuousValueLayer** (const ContinuousValueLayer< ValueType, Borders > &other)
- ContinuousValueLayer & **operator=** (const ContinuousValueLayer< ValueType, Borders > &rhs)
- ContinuousValueLayer (const std::string &name, const GridDimensions &dimensions, const ValueType &defaultValue=ValueType())

  Creates a ContinuousValueLayer whose cells contain a default value of ValueType() with the specified dimensions.
- ValueType & get (const Point< double > &pt)

  Gets the value at the specified point.
- void set (const ValueType &value, const Point< double > &pt)

  Sets the value at the specified point.

5.22.1 Detailed Description

template<typename ValueType, typename Borders>class repast::ContinuousValueLayer< ValueType, Borders >

Continous value layer whose location coordinates are double.

Template Parameters

ValueType	the type of what the value layer stores.
Borders	the type of borders (wrapped / periodic, strict). Border types can be found in GridComponents.h

5.22.2 Constructor & Destructor Documentation

5.22.2.1 template<typename ValueType , typename Borders > repast::ContinuousValueLayer< ValueType, Borders >::ContinuousValueLayer (const std::string & *name,* const GridDimensions & *dimensions,* const ValueType & *defaultValue* = `ValueType()`)

Creates a ContinuousValueLayer whose cells contain a default value of ValueType() with the specified dimensions.

Parameters

name	the name of the ContinuousValueLayer
dimension	the dimensions of the ContinuousValueLayer
dense	whether or not the ValueLayer will be densely populated or not
defaultValue	the default value to return if no value has been set of a location. The default is the result of ValueType().

5.22.3 Member Function Documentation

5.22.3.1 template<typename ValueType , typename Borders > ValueType & repast::ContinuousValueLayer< ValueType, Borders >::get (const Point< double > & *pt*) [virtual]

Gets the value at the specified point.

If no value has been set at the specified point then this returns the default value.

param pt the location to get the value of

Returns

the value at the specified point, or if no value has been set, then the default value.

Implements repast::ValueLayer< ValueType, double >.

5.22.3.2 template<typename ValueType , typename Borders > void repast::ContinuousValueLayer< ValueType, Borders >::set (const ValueType & *value*, const Point< double > & *pt*) [virtual]

Sets the value at the specified point.

Parameters

value	the value
pt	the point where the value should be stored

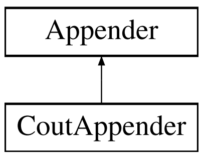
Implements repast::ValueLayer< ValueType, double >.

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

- repast_hpc/ValueLayer.h

5.23 CoutAppender Class Reference

Inheritance diagram for CoutAppender:



Public Member Functions

- void **write** (const string &line)

Additional Inherited Members

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

- repast_hpc/logger.cpp

5.24 repast::data_type_traits< T > Struct Template Reference

Base class for specialized int and double type classes.

`#include <SVDataSource.h>`

5.24.1 Detailed Description

template<typename T>struct repast::data_type_traits< T >

Base class for specialized int and double type classes.

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

- repast_hpc/SVDataSource.h

5.25 repast::data_type_traits< double > Struct Template Reference

Double data types for SVDataSource objects.

`#include <SVDataSource.h>`

Static Public Member Functions

- static SVDataSource::DataType **data_type** ()

5.25.1 Detailed Description

template<>struct repast::data_type_traits< double >

Double data types for SVDataSource objects.

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

- repast_hpc/SVDataSource.h

5.26 repast::data_type_traits< int > Struct Template Reference

Int data types for SVDataSource objects.

`#include <SVDataSource.h>`

Static Public Member Functions

- static SVDataSource::DataType **data_type** ()

5.26.1 Detailed Description

template<>struct repast::data_type_traits< int >

Int data types for SVDataSource objects.

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

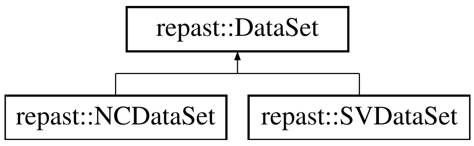
- repast_hpc/SVDataSource.h

5.27 repast::DataSet Class Reference

Interface for recording and writing data.

```
#include <DataSet.h>
```

Inheritance diagram for repast::DataSet:



Public Member Functions

- virtual void record ()=0

  Records the data.
- virtual void write ()=0

  Writes the data.
- virtual void close ()=0

  Closes the dataset, after which it must be recreated to be used.

5.27.1 Detailed Description

Interface for recording and writing data.

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

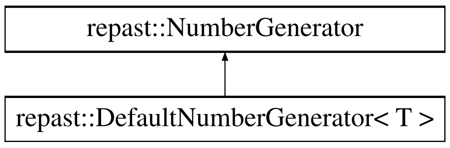
- repast_hpc/DataSet.h

5.28 repast::DefaultNumberGenerator< T > Class Template Reference

Adapts the templated boost::variate_generator to the NumberGenerator interface.

```
#include <Random.h>
```

Inheritance diagram for repast::DefaultNumberGenerator< T >:



Public Member Functions

- **DefaultNumberGenerator** (T generator)
- double next ()

    Gets the "next" number from this Number Generator.

5.28.1 Detailed Description

template<typename T>class repast::DefaultNumberGenerator< T >

Adapts the templated boost::variate_generator to the NumberGenerator interface.

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

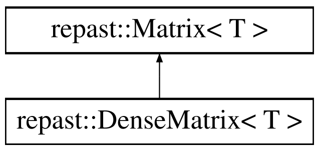
- repast_hpc/Random.h

5.29 repast::DenseMatrix< T > Class Template Reference

A dense matrix implementation that stores each cell individually.

```
#include <matrix.h>
```

Inheritance diagram for repast::DenseMatrix< T >:



Public Member Functions

- DenseMatrix (const DenseMatrix< T > &)

    Creates a DenseMatrix as a copy of the specified DenseMatrix.
- DenseMatrix< T > & **operator=** (const DenseMatrix< T > &)
- DenseMatrix (const Point< int > &shape, const T &defValue=T())

    Creates a DenseMatrix of the specified shape and default value.
- T & get (const Point< int > &index)

    Gets the value at the specified index.
- void set (const T &value, const Point< int > &index)

    Sets the value at the specified index.

Additional Inherited Members

5.29.1 Detailed Description

template<typename T>class repast::DenseMatrix< T >

A dense matrix implementation that stores each cell individually.

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

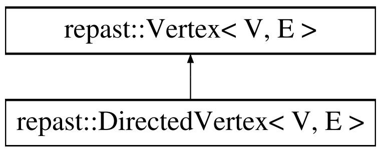
- repast_hpc/matrix.h

5.30 repast::DirectedVertex< V, E > Class Template Reference

Used internally by repast graphs / networks to encapsulate the vertices of a directed graph.

```
#include <DirectedVertex.h>
```

Inheritance diagram for repast::DirectedVertex< V, E >:



Public Member Functions

- DirectedVertex (boost::shared_ptr< V > item)

  Creates a DirectedVertex that will contain the specified item.
- virtual boost::shared_ptr< E > removeEdge (Vertex< V, E > ∗other, EdgeType type)

  Removes the edge of the specified type between this Vertex and the specified Vertex.
- virtual boost::shared_ptr< E > findEdge (Vertex< V, E > ∗other, EdgeType type)

  Finds the edge of the specified type between this Vertex and the specified vertex.
- virtual void addEdge (Vertex< V, E > ∗other, boost::shared_ptr< E > edge, EdgeType type)

  Adds an edge of the specified type between this Vertex and the specified vertex.
- virtual void successors (std::vector< V ∗ > &out)

  Gets the successors of this Vertex.
- virtual void predecessors (std::vector< V ∗ > &out)

  Gets the predecessors of this Vertex.
- virtual void adjacent (std::vector< V ∗ > &out)

  Gets the Vertices adjacent to this Vertex.
- virtual void edges (EdgeType type, std::vector< boost::shared_ptr< E > > &out)

  Gets all the edges of the specified type in which this Vertex participates and return them in out.
- int inDegree ()

  Gets the in degree of this Vertex.
- int outDegree ()

  Gets the out degree of this Vertex.

5.30.1 Detailed Description

template<typename V, typename E>class repast::DirectedVertex< V, E >

Used internally by repast graphs / networks to encapsulate the vertices of a directed graph.

Template Parameters

V	the type of object stored by in a Vertex.
E	the EdgeType of the network.

5.30.2 Member Function Documentation

5.30.2.1 template<typename V , typename E > void repast::DirectedVertex< V, E >::addEdge (Vertex< V, E > ∗ *other,* boost::shared_ptr< E > *edge,* EdgeType *type*) [virtual]

Adds an edge of the specified type between this Vertex and the specified vertex.

Parameters

edge	the edge to add
other	the other end of the edge
type	the type of edge to add

Implements repast::Vertex< V, E >.

5.30.2.2 template<typename V , typename E > void **repast::DirectedVertex**< V, E >::adjacent (std::vector< V ∗ > & *out*) [virtual]

Gets the Vertices adjacent to this Vertex.

Parameters

out	*the*	vector where the adjacent vectors will be put

Implements repast::Vertex< V, E >.

5.30.2.3 template<typename V , typename E > void **repast::DirectedVertex**< V, E >::edges (**EdgeType** *type*, std::vector< boost::shared_ptr< E > > & *out*) [virtual]

Gets all the edges of the specified type in which this Vertex participates and return them in out.

Parameters

	type	the type of edges to get
out	*where*	the edges will be put.

Implements repast::Vertex< V, E >.

5.30.2.4 template<typename V , typename E > boost::shared_ptr< E > **repast::DirectedVertex**< V, E >::findEdge (**Vertex**< V, E > ∗ *other*, **EdgeType** *type*) [virtual]

Finds the edge of the specified type between this Vertex and the specified vertex.

Parameters

other	the other end of the edge
type	the type of edge to remove

Returns

the found edge, or 0.

Implements repast::Vertex< V, E >.

5.30.2.5 template<typename V , typename E > int **repast::DirectedVertex**< V, E >::inDegree ( ) [virtual]

Gets the in degree of this Vertex.

Returns

the in degree of this Vertex.

Implements repast::Vertex< V, E >.

5.30.2.6 template<typename V , typename E > int repast::DirectedVertex< V, E >::outDegree ( ) [virtual]

Gets the out degree of this Vertex.

Returns

the out degree of this Vertex.

Implements repast::Vertex< V, E >.

5.30.2.7 template<typename V , typename E > void repast::DirectedVertex< V, E >::predecessors (std::vector< V ∗ > & *out*) [virtual]

Gets the predecessors of this Vertex.

Parameters

out	*the*	vector where any predecessors will be put

Implements repast::Vertex< V, E >.

5.30.2.8 template<typename V , typename E > boost::shared_ptr< E > repast::DirectedVertex< V, E >::removeEdge (Vertex< V, E > ∗ *other*, EdgeType *type*) [virtual]

Removes the edge of the specified type between this Vertex and the specified Vertex.

Parameters

other	the other end of the edge
type	the type of edge to remove

Returns

the removed edge if such an edge was found, otherwise 0.

Implements repast::Vertex< V, E >.

5.30.2.9 template<typename V , typename E > void repast::DirectedVertex< V, E >::successors (std::vector< V ∗ > & *out*) [virtual]

Gets the successors of this Vertex.

Parameters

out	*the*	vector where any successors will be put

Implements repast::Vertex< V, E >.

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

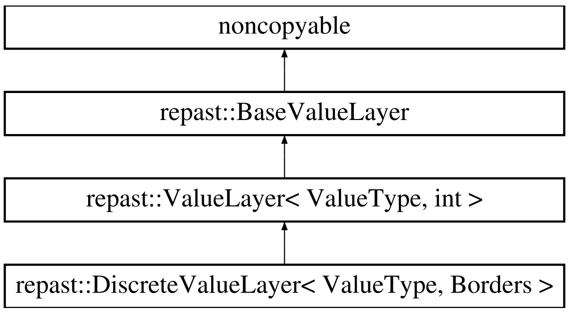
- repast_hpc/DirectedVertex.h

5.31 repast::DiscreteValueLayer< ValueType, Borders > Class Template Reference

Creates ValueLayer whose location coordinates are ints.

`#include <ValueLayer.h>`

Inheritance diagram for repast::DiscreteValueLayer< ValueType, Borders >:


noncopyable
repast::BaseValueLayer
repast::ValueLayer< ValueType, int >
repast::DiscreteValueLayer< ValueType, Borders >

Public Member Functions

- **DiscreteValueLayer** (const DiscreteValueLayer< ValueType, Borders > &other)
- DiscreteValueLayer & **operator=** (const DiscreteValueLayer< ValueType, Borders > &rhs)
- DiscreteValueLayer (const std::string &name, const GridDimensions &dimensions, bool dense, const ValueType &defaultValue=ValueType())

  Creates a DiscreteValueLayer whose cells contain a default value of ValueType() with the specified dimensions.
- ValueType & get (const Point< int > &pt)

  Gets the value at the specified point.
- void set (const ValueType &value, const Point< int > &pt)

  Sets the value at the specified point.

5.31.1 Detailed Description

template<typename ValueType, typename Borders>class repast::DiscreteValueLayer< ValueType, Borders >

Creates ValueLayer whose location coordinates are ints.

Template Parameters

ValueType	the type of what the value layer stores.
Borders	the type of borders (wrapped / periodic, strict). Border types can be found in GridComponents.h

5.31.2 Constructor & Destructor Documentation

5.31.2.1 template<typename ValueType , typename Borders > repast::DiscreteValueLayer< ValueType, Borders >::DiscreteValueLayer (const std::string & *name,* const GridDimensions & *dimensions,* bool *dense,* const ValueType & *defaultValue* = `ValueType()`)

Creates a DiscreteValueLayer whose cells contain a default value of ValueType() with the specified dimensions.

Parameters

name	the name of the DiscreteValueLayer
dimension	the dimensions of the DiscreteValueLayer
dense	whether or not the ValueLayer will be densely populated or not
defaultValue	the default value to return if no value has been set of a location. The default is the result of ValueType().

5.31.3 Member Function Documentation

5.31.3.1 template<typename ValueType , typename Borders > ValueType & **repast::DiscreteValueLayer**< ValueType, **Borders** >::get (const **Point**< int > & *pt*) `[virtual]`

Gets the value at the specified point.

If no value has been set at the specified point then this returns the default value.

param pt the location to get the value of

Returns

the value at the specified point, or if no value has been set, then the default value.

Implements repast::ValueLayer< ValueType, int >.

5.31.3.2 template<typename ValueType , typename Borders > void **repast::DiscreteValueLayer**< ValueType, **Borders** >::set (const ValueType & *value*, const **Point**< int > & *pt*) `[virtual]`

Sets the value at the specified point.

Parameters

value	the value
pt	the point where the value should be stored

Implements repast::ValueLayer< ValueType, int >.

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

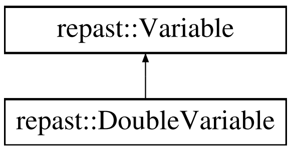
- repast_hpc/ValueLayer.h

5.32 repast::DoubleVariable Class Reference

Used in SVDataSet to manage double data.

`#include <Variable.h>`

Inheritance diagram for repast::DoubleVariable:



Public Member Functions

- virtual void write (size_t index, std::ofstream &out)

  Writes the data at the specified index to the specified ofstream.
- virtual void insert (double ∗array, size_t size)

  Inserts all the doubles in the double array into the collection of data stored in this Variable.
- virtual void insert (int ∗array, size_t size)

  Inserts all the ints in the int array into the collection of data stored in this Variable.
- virtual void clear ()

  Clears this Variable of all the data stored in it.

5.32.1 Detailed Description

Used in SVDataSet to manage double data.

5.32.2 Member Function Documentation

5.32.2.1 void repast::DoubleVariable::insert (double ∗ *array*, size_t *size*) [virtual]

Inserts all the doubles in the double array into the collection of data stored in this Variable.

Parameters

array	the array to insert
size	the size of the array

Implements repast::Variable.

5.32.2.2 void repast::DoubleVariable::insert (int ∗ *array*, size_t *size*) [virtual]

Inserts all the ints in the int array into the collection of data stored in this Variable.

Parameters

array	the array to insert
size	the size of the array

Implements repast::Variable.

5.32.2.3 void repast::DoubleVariable::write (size_t *index*, std::ofstream & *out*) [virtual]

Writes the data at the specified index to the specified ofstream.

Parameters

index	the index of the data to write
out	the ofstream to write the data to

Implements repast::Variable.

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

- repast_hpc/Variable.h
- repast_hpc/Variable.cpp

5.33 repast::EdgeExporter< E > Class Template Reference

DEPRECATED Handles exporting edges created locally between one or more non-local agents.

```
#include <SharedNetwork.h>
```

Public Types

- typedef std::map< int,
  std::vector< boost::shared_ptr
  < E > > ∗ >::iterator **EdgeMapIterator**

Public Member Functions

- void **addAgentExportRequest** (int exportTo, const AgentId &id)
- void edgeRemoved (boost::shared_ptr< E > edge, std::map< int, std::vector< std::pair< AgentId, AgentId > > > &removeMap)

  Whether or not this is exported the specified edge.
- void addEdge (boost::shared_ptr< E > edge)

  Tests if the edge needs to be exported and if so adds it to the collection of edges to be exported.
- void gatherReceivers (std::vector< int > &out)

  Gathers the receivers into out.
- void gatherExporters (std::vector< int > &out)

  Gathers the procs that this will send export requests to into out.
- void sendExportRequests (boost::mpi::communicator &comm, std::vector< boost::mpi::request > &requests)

  Send the export requests.
- std::map< int, std::vector
  < boost::shared_ptr< E > > ∗ > & getEdgesToExport ()

  Gets the edges to export.
- std::map< int, std::vector
  < boost::shared_ptr< E > > ∗ > & getExportedEdges ()

  Gets the edges this process is exporting.
- void cleanUp ()

  Cleans up after exported edges have been sent and received.

Friends

- template<typename Vertex , typename Edge , typename AgentContent , typename EdgeContent , typename EdgeManager , typename AgentCreator >
  void createComplementaryEdges (SharedNetwork< Vertex, Edge, EdgeContent, EdgeManager > ∗net, SharedContext< Vertex > &context, EdgeManager &edgeManager, AgentCreator &creator)

  Notifies other processes of any edges that have been created between nodes on this process and imported nodes.

5.33.1 Detailed Description

template<typename E>class repast::EdgeExporter< E >

DEPRECATED Handles exporting edges created locally between one or more non-local agents.

This also coordinates notification of which processes should be exporting to which in the case of edges where a node is foreign to the sending and receiving process.

All this is done internally in the SharedNetwork.

Deprecated As of Version 2.0 replaced by ProjectionInfoPacket

5.33.2 Member Function Documentation

5.33.2.1 template<typename E > void repast::EdgeExporter< E >::gatherReceivers (std::vector< int > & *out*)

Gathers the receivers into out.

A receiver is a process this EdgeExporter should send an edge to.

5.33.2.2 template< typename E > void **repast::EdgeExporter**< E >::sendExportRequests (boost::mpi::communicator & *comm*, std::vector< boost::mpi::request > & *requests*)

Send the export requests.

This does an isend and the resulting requests are placed in the specified vector.

5.33.3 Friends And Related Function Documentation

5.33.3.1 template<typename E> template<typename Vertex , typename Edge , typename AgentContent , typename EdgeContent , typename EdgeManager , typename AgentCreator > void createComplementaryEdges (**SharedNetwork**< Vertex, Edge, EdgeContent, EdgeManager > ∗ *net*, **SharedContext**< Vertex > & *context*, EdgeManager & *edgeManager*, AgentCreator & *creator*) `[friend]`

Notifies other processes of any edges that have been created between nodes on this process and imported nodes.

The other process will then create the complimentary edge. For example, if P1 creates an edge between A and B where B resides on P2, then this method will notify P2 to create the incoming edge A->B on its copy of B. Any unknown agents will be added to the context. For example, if P2 didn't have a reference to A, then A will be added to P2's context.

Parameters

net	the network in which to create the complementary edges or from which to send complementary edges
context	the context that contains the agents in the process
edgeManager	creates edges from EdgeContent and creates EdgeContent from an edge and a context.
creator	creates agents from AgentContent.

Template Parameters

Vertex	the vertex (agent) type
Edge	the edge type
AgentContent	the serializable struct or class that describes the agent state. It must contain a getId() method that returns the AgentId of the agent it describes.
EdgeContent	the serializable struct or class that describes edge state. At the very least EdgeContent must contain two public fields sourceContent and targetContent of type AgentContent. These represent the source and target of the edge.
EdgeManager	create edges from EdgeContent and provides EdgeContent given a context and an edge of type Edge. It must implement void provideEdgeContent(constEdge∗ edge, std::vector<EdgeContent>& edgeContent) and Edge∗ createEdge(repast::Context<Vertex>& context, EdgeContent& edge);
AgentCreator	creates agents from AgentContent, implementing the following method Vertex∗ createAgent(constAgentContent& content);

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

- repast_hpc/SharedNetwork.h

5.34 repast::EventCompare Class Reference

Compares ScheduledEvents based on their tick times.

```
#include <Schedule.h>
```

Public Member Functions

- int **operator()** (const ScheduledEvent ∗one, const ScheduledEvent ∗two)

5.34.1 Detailed Description

Compares ScheduledEvents based on their tick times.

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

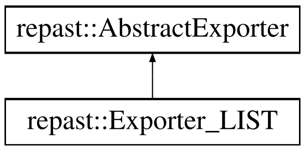
- repast_hpc/Schedule.h

5.35 repast::Exporter_LIST Class Reference

Maintains a list of agents being exported for each receiving process.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::Exporter_LIST:



Public Member Functions

- virtual void registerIncomingRequests (std::vector< AgentRequest > &requests)

    Makes a record of the data receives (in the form of a vector of AgentRequests) so that the agents' data can be sent to the requesting processes.
- **Exporter_LIST** (StatusMap ∗outgoingStatusMap, AgentExporterData ∗outgoingAgentExporterInfo)
- virtual std::string getReport ()

    Gets a printable report of the state of this object.

Additional Inherited Members

5.35.1 Detailed Description

Maintains a list of agents being exported for each receiving process.

An agent that is requested more than once will appear on this list more than once; canceling an agent just once removes only one of its appearances on this list. A 'send' will be created for a receiving process only if that process has entries in the list.

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

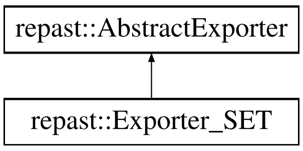
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.36 repast::Exporter_SET Class Reference

Maintains a set of agents being exported for each receiving process.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::Exporter_SET:



Public Member Functions

- virtual void registerIncomingRequests (std::vector< AgentRequest > &requests)

  Makes a record of the data receives (in the form of a vector of AgentRequests) so that the agents' data can be sent to the requesting processes.
- **Exporter_SET** (StatusMap ∗outgoingStatusMap, AgentExporterData ∗outgoingAgentExporterInfo)
- virtual std::string getReport ()

  Gets a printable report of the state of this object.

Additional Inherited Members

5.36.1 Detailed Description

Maintains a set of agents being exported for each receiving process.

An agent that is requested more than once will appear in this set only once; canceling an agent just once removes it from this set. A 'send' will be created for a receiving process only if that process has entries in the list.

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

- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.37 repast::ExportRequest Class Reference

DEPRECATED Used to send a request for agent information from another process

```
#include <SharedNetwork.h>
```

Public Member Functions

- **ExportRequest** (int exportTo, AgentId id)
- AgentId **agent** () const
- int **exportTo** ()

Friends

- class **boost::serialization::access**

5.37.1 Detailed Description

DEPRECATED Used to send a request for agent information from another process

Deprecated As of Version 2.0

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

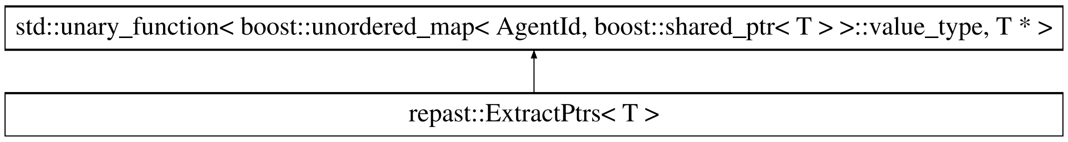
- repast_hpc/SharedNetwork.h
- repast_hpc/SharedNetwork.cpp

5.38 repast::ExtractPtrs< T > Struct Template Reference

Unary function that allows retrieving the occupants of locations.

`#include <MultipleOccupancy.h>`

Inheritance diagram for repast::ExtractPtrs< T >:


std::unary_function< boost::unordered_map< AgentId, boost::shared_ptr< T > >::value_type, T * >
repast::ExtractPtrs< T >

Public Member Functions

- T $*$ **operator()** (typename boost::unordered_map< AgentId, boost::shared_ptr< T > >::value_type &val)

5.38.1 Detailed Description

template<typename T>struct repast::ExtractPtrs< T >

Unary function that allows retrieving the occupants of locations.

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

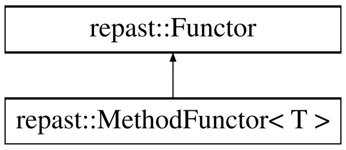
- repast_hpc/MultipleOccupancy.h

5.39 repast::Functor Class Reference

Functor interface.

`#include <Schedule.h>`

Inheritance diagram for repast::Functor:



Public Member Functions

- virtual void **operator()** ()=0

5.39.1 Detailed Description

Functor interface.

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

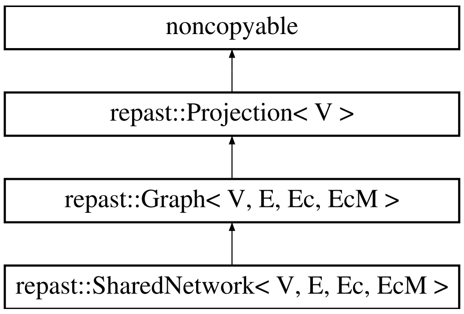
- repast_hpc/Schedule.h
- repast_hpc/Schedule.cpp

5.40 repast::Graph< V, E, Ec, EcM > Class Template Reference

Graph / Network implementation where agents are vertices in the graph.

`#include <Graph.h>`

Inheritance diagram for repast::Graph< V, E, Ec, EcM >:


noncopyable
repast::Projection< V >
repast::Graph< V, E, Ec, EcM >
repast::SharedNetwork< V, E, Ec, EcM >

Public Types

- typedef
  boost::transform_iterator
  < NodeGetter< V, E >, typename
  VertexMap::const_iterator > vertex_iterator

  An iterator over the agents that are the vertices in this Graph.

Public Member Functions

- Graph (std::string name, bool directed, EcM ∗edgeContentMgr)

  Creates a Graph with the specified name.
- Graph (const Graph< V, E, Ec, EcM > &graph)

  Copy constructor for the graph.
- Graph & **operator=** (const Graph &graph)
- virtual boost::shared_ptr< E > addEdge (V ∗source, V ∗target)

  Adds an edge between source and target to this Graph.
- virtual boost::shared_ptr< E > addEdge (V ∗source, V ∗target, double weight)

  Adds an edge with the specified weight between source and target to this Graph.
- virtual boost::shared_ptr< E > findEdge (V ∗source, V ∗target)

  Gets the edge between the source and target or 0 if no such edge is found.
- virtual void successors (V ∗vertex, std::vector< V ∗ > &out)

  Gets the sucessors of the specified vertex and puts them in out.
- virtual void predecessors (V ∗vertex, std::vector< V ∗ > &out)

  Gets the predecessors of the specified vertex and puts them in out.
- virtual void adjacent (V ∗vertex, std::vector< V ∗ > &out)

  Gets all the agent adjacent to the specified vertex.
- virtual void removeEdge (V ∗source, V ∗target)

  Removes the edge between source and target from this Graph.
- virtual void removeEdge (const AgentId &source, const AgentId &target)

Removes the edge between source and target from this Graph.

- virtual int inDegree (V ∗vertex)

  Gets the in-degree of the specified vertex.
- virtual int outDegree (V ∗vertex)

  Gets the out-degree of the specified vertex.
- int edgeCount () const

  Gets the number of edges in this Graph.
- int vertexCount () const

  Gets the number of vertices in this Graph.
- vertex_iterator verticesBegin ()

  Gets the start of an iterator over all the vertices in this graph.
- vertex_iterator verticesEnd ()

  Gets the end of an iterator over all the vertices in this graph.
- void **showEdges** ()
- virtual bool **isMaster** (E ∗e)=0
- virtual bool keepsAgentsOnSyncProj ()

  Should return true if the Projection implemented can 'keep' some (non-local) agents during a projection information synchronization operation.
- virtual bool sendsSecondaryAgentsOnStatusExchange ()

  Should return true if the Projection implemented will send secondary agents during a status exchange.
- virtual void getInfoExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceiveFrom)

  Gets the set of processes with which this Projection exchanges projection info.
- virtual void getAgentStatusExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceive-From)

  Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.
- virtual void getProjectionInfo (std::vector< AgentId > &agents, std::vector< ProjectionInfoPacket ∗ > &pack-ets, bool secondaryInfo=false, std::set< AgentId > ∗secondaryIds=0, int destProc=-1)

  Convenience wrapper that gets all of the projection information for the agents specified (calls implementation in child class that gets only the information for one agent).
- virtual ProjectionInfoPacket ∗ **getProjectionInfo** (AgentId id, bool secondaryInfo=false, std::set< AgentId > ∗secondaryIds=0, int destProc=-1)
- virtual void **updateProjectionInfo** (ProjectionInfoPacket ∗pip, Context< V > ∗context)
- virtual void **getRequiredAgents** (std::set< AgentId > &agentsToTest, std::set< AgentId > &agents-Required, RADIUS radius=Projection< V >::PRIMARY)
- virtual void getAgentsToPush (std::set< AgentId > &agentsToTest, std::map< int, std::set< AgentId > > &agentsToPush)

  Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.
- virtual void **cleanProjectionInfo** (std::set< AgentId > &agentsToKeep)
- void **clearConflictedEdges** ()
- void **getConflictedEdges** (std::set< boost::shared_ptr< E > > &conflictedEdges)

Public Attributes

- std::set< int > **ranksToSendProjInfoTo**
- std::set< int > **ranksToReceiveProjInfoFrom**
- std::set< int > **ranksToSendAgentStatusInfoTo**
- std::set< int > **ranksToReceiveAgentStatusInfoFrom**
- bool **keepsAgents**
- bool **sendsSecondaryAgents**

Protected Types

- typedef boost::unordered_map
  < AgentId, Vertex< V, E >
  ∗, HashId > **VertexMap**
- typedef VertexMap::iterator **VertexMapIterator**
- typedef Projection< V >::RADIUS **RADIUS**

Protected Member Functions

- void **cleanUp** ()
- void **init** (const Graph &graph)
- virtual bool **addAgent** (boost::shared_ptr< V > agent)
- virtual void **removeAgent** (V ∗agent)
- virtual void **doAddEdge** (boost::shared_ptr< E > edge, bool allowOverwrite=true)

Protected Attributes

- int **edgeCount_**
- bool **isDirected**
- VertexMap **vertices**
- EcM ∗ **edgeContentManager**

5.40.1 Detailed Description

template<typename V, typename E, typename Ec, typename EcM>class repast::Graph< V, E, Ec, EcM >

Graph / Network implementation where agents are vertices in the graph.

Template Parameters

V	the type agents in the graph. This type should extend repast::Agent
E	the edge type of the graph. This type should extend repast::RepastEdge.
Ec	class of serializable Edge Content
EcM	Class that is capable of transforming an Edge into Edge Content and vice versa

5.40.2 Constructor & Destructor Documentation

5.40.2.1 template<typename V, typename E, typename Ec, typename EcM> repast::Graph< V, E, Ec, EcM >::Graph (std::string *name*, bool *directed*, EcM ∗ *edgeContentMgr*) `[inline]`

Creates a Graph with the specified name.

Parameters

name	the name of the graph
directed	whether or not the created Graph is directed

5.40.3 Member Function Documentation

5.40.3.1 template<typename V , typename E , typename Ec , typename EcM > boost::shared_ptr< E > repast::Graph< V, E, Ec, EcM >::addEdge (V ∗ *source*, V ∗ *target*) `[virtual]`

Adds an edge between source and target to this Graph.

Parameters

source	the source of the edge
target	the target of the edge

Returns

the added edge.

5.40.3.2 template<typename V , typename E , typename Ec , typename EcM > boost::shared_ptr< E > **repast::Graph**< V, E, Ec, EcM >::addEdge (V ∗ *source*, V ∗ *target*, double *weight*) `[virtual]`

Adds an edge with the specified weight between source and target to this Graph.

Parameters

source	the source of the edge
target	the target of the edge
weight	the weight of the edge

Returns

the added edge.

5.40.3.3 template<typename V , typename E , typename Ec , typename EcM > void **repast::Graph**< V, E, Ec, EcM >::adjacent (V ∗ *vertex*, std::vector< V ∗ > & *out*) `[virtual]`

Gets all the agent adjacent to the specified vertex.

Parameters

	vertex	the vertex whose adjacent agents we want to get
`out`	*the*	vector where the results will be put

5.40.3.4 template<typename V, typename E, typename Ec, typename EcM> int **repast::Graph**< V, E, Ec, EcM >::edgeCount ( ) const `[inline]`

Gets the number of edges in this Graph.

Returns

the number of edges in this Graph.

5.40.3.5 template<typename V , typename E , typename Ec , typename EcM > boost::shared_ptr< E > **repast::Graph**< V, E, Ec, EcM >::findEdge (V ∗ *source*, V ∗ *target*) `[virtual]`

Gets the edge between the source and target or 0 if no such edge is found.

Parameters

source	the source of the edge to find

target	the target of the edge to find

Returns

the found edge or 0.

5.40.3.6 template<typename V , typename E , typename Ec , typename EcM > void repast::Graph< V, E, Ec, EcM >::getAgentStatusExchangePartners (std::set< int > & *psToSendTo,* std::set< int > & *psToReceiveFrom*) [virtual]

Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.

In the most general case this will be all other processors. However, simulations where agents move in spaces will usually exchange agents only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implements repast::Projection< V >.

5.40.3.7 template<typename V , typename E , typename Ec , typename EcM > void repast::Graph< V, E, Ec, EcM >::getAgentsToPush (std::set< AgentId > & *agentsToTest,* std::map< int, std::set< AgentId > > & *agentsToPush*) [virtual]

Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.

Generally spaces must push agents that are in 'buffer zones' and graphs must push local agents that are vertices to master edges where the other vertex is non- local. The results are returned per-process in the agentsToPush map.

Implements repast::Projection< V >.

5.40.3.8 template<typename V , typename E , typename Ec , typename EcM > void repast::Graph< V, E, Ec, EcM >::getInfoExchangePartners (std::set< int > & *psToSendTo,* std::set< int > & *psToReceiveFrom*) [virtual]

Gets the set of processes with which this Projection exchanges projection info.

In the most general case this will be all other processors; this is the case for graphs, where agent connections can be arbitrary. However, spaces usually exchange information only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implements repast::Projection< V >.

5.40.3.9 template<typename V , typename E , typename Ec , typename EcM > int repast::Graph< V, E, Ec, EcM >::inDegree (V * *vertex*) [virtual]

Gets the in-degree of the specified vertex.

Returns

the in-degree of the specified vertex.

5.40.3.10 template<typename V, typename E, typename Ec, typename EcM> virtual bool repast::Graph< V, E, Ec, EcM >::keepsAgentsOnSyncProj ( ) [inline], [virtual]

Should return true if the Projection implemented can 'keep' some (non-local) agents during a projection information synchronization operation.

Generally spaces will allow all non-local agents to be deleted, but graphs keep the non-local agents that participate in Master edges.

It is possible to override these. A graph projection can be created that does not permit non-local agents to be 'kept'. This would be an extremely unusual use case, but it is possible.

Note that these are used for optimization. If no projection in a given context keeps any agents, several steps in the synchronization algorithm can be omitted. Of course, omitting these steps when a projection actually retains agents can caused undefined problems.

Returns

true if this projection will keep non-local agents during a projection information synchronziation event, false if it will not.

Implements repast::Projection< V >.

5.40.3.11 template<typename V , typename E , typename Ec , typename EcM > int **repast::Graph**< V, E, Ec, EcM >::outDegree (V ∗ *vertex*) `[virtual]`

Gets the out-degree of the specified vertex.

Returns

the out-degree of the specified vertex.

5.40.3.12 template<typename V , typename E , typename Ec , typename EcM > void **repast::Graph**< V, E, Ec, EcM >::predecessors (V ∗ *vertex*, std::vector< V ∗ > & *out*) `[virtual]`

Gets the predecessors of the specified vertex and puts them in out.

Parameters

	vertex	the vertex whose predecessors we want to get
`out`	*where*	the predecessors will be returned

5.40.3.13 template<typename V , typename E , typename Ec , typename EcM > void **repast::Graph**< V, E, Ec, EcM >::removeEdge (V ∗ *source*, V ∗ *target*) `[virtual]`

Removes the edge between source and target from this Graph.

Parameters

source	the source of the edge
target	the target of the edge

Reimplemented in repast::SharedNetwork< V, E, Ec, EcM >.

5.40.3.14 template<typename V , typename E , typename Ec , typename EcM > void **repast::Graph**< V, E, Ec, EcM >::removeEdge (const AgentId & *source*, const AgentId & *target*) `[virtual]`

Removes the edge between source and target from this Graph.

Parameters

source	the id of the vertex that is the source of the edge
target	the id of the vertex that is the target of the edge

5.40.3.15 template<typename V, typename E, typename Ec, typename EcM> virtual bool repast::Graph< V, E, Ec, EcM >::sendsSecondaryAgentsOnStatusExchange ( ) [inline], [virtual]

Should return true if the Projection implemented will send secondary agents during a status exchange.

Generally spaces do not and graphs do.

If no secondary agents will be sent, portions of the algorithm can be omitted for optimization.

Returns

true if the Projection returns secondary agents, false if not

Implements repast::Projection< V >.

5.40.3.16 template<typename V , typename E , typename Ec , typename EcM > void repast::Graph< V, E, Ec, EcM >::successors (V ∗ *vertex*, std::vector< V ∗ > & *out*) [virtual]

Gets the sucessors of the specified vertex and puts them in out.

Parameters

	vertex	the vertex whose successors we want to get
`out`	*where*	the successors will be returned

5.40.3.17 template<typename V, typename E, typename Ec, typename EcM> int repast::Graph< V, E, Ec, EcM >::vertexCount ( ) const [inline]

Gets the number of vertices in this Graph.

Returns

the number of vertices in this Graph.

5.40.3.18 template<typename V, typename E, typename Ec, typename EcM> vertex_iterator repast::Graph< V, E, Ec, EcM >::verticesBegin ( ) [inline]

Gets the start of an iterator over all the vertices in this graph.

The iterator dereferences to a pointer to agents of type V.

Returns

the start of an iterator over all the vertices in this graph.

5.40.3.19 template<typename V, typename E, typename Ec, typename EcM> vertex_iterator repast::Graph< V, E, Ec, EcM >::verticesEnd ( ) [inline]

Gets the end of an iterator over all the vertices in this graph.

The iterator dereferences to a pointer to agents of type V.

Returns

the end of an iterator over all the vertices in this graph.

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

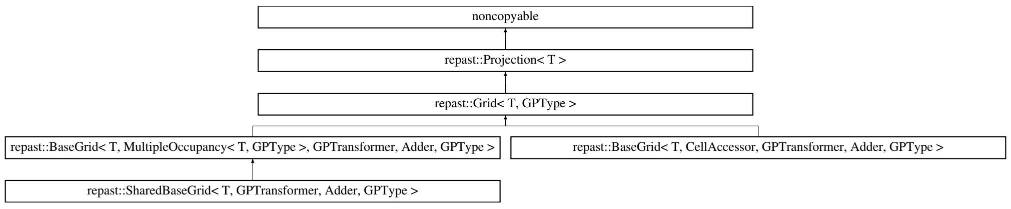
- repast_hpc/Graph.h

5.41 repast::Grid< T, GPType > Class Template Reference

Abstract interface for Grids and ContinuousSpaces.

`#include <Grid.h>`

Inheritance diagram for repast::Grid< T, GPType >:


noncopyable
repast::Projection< T >
repast::Grid< T, GPType >
repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >
repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >
repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >

Public Member Functions

- Grid (std::string name)

  Creates a Grid with the specified name.
- virtual bool contains (const AgentId &id)=0

  Gets whether or not this grid contains the agent with the specified id.
- virtual bool moveTo (const AgentId &id, const Point< GPType > &pt)=0

  Moves the specified agent to the specified point.
- virtual std::pair< bool, Point
  < GPType > > moveByVector (const T ∗agent, double distance, const std::vector< double > &anglesIn-Radians)=0

  Moves the specifed object the specified distance from its current position along the specified angle.
- virtual std::pair< bool, Point
  < GPType > > moveByDisplacement (const T ∗agent, const std::vector< GPType > &displacement)=0

  Moves the specified object from its current location by the specified amount.
- virtual const GridDimensions dimensions () const =0

  Gets the dimensions of this Grid.
- virtual T ∗ getObjectAt (const Point< GPType > &pt) const =0

  Gets the first object found at the specified point, or NULL if there is no such object.
- virtual void getObjectsAt (const Point< GPType > &pt, std::vector< T ∗ > &out) const =0

  Gets all the objects found at the specified point.
- virtual bool getLocation (const T ∗agent, std::vector< GPType > &out) const =0

  Gets the location of this agent and puts it in the specified vector.
- virtual bool getLocation (const AgentId &id, std::vector< GPType > &out) const =0

  Gets the location of this agent and puts it in the specified vectors.
- virtual void getDisplacement (const Point< GPType > &pt1, const Point< GPType > &pt2, std::vector< GPType > &out) const =0

  Gets vector difference between point 1 and point 2, putting the result in out.
- virtual double getDistance (const Point< GPType > &pt1, const Point< GPType > &pt2) const =0

Gets the distance between the two grid points.

- virtual double getDistanceSq (const Point< GPType > &pt1, const Point< GPType > &pt2) const =0

  Gets the square of the distance between the two grid points.
- virtual void translate (const Point< GPType > &location, const Point< GPType > &displacement, std::vector< GPType > &out) const =0

  Translates the specified location by the specified displacement put the result in out.
- virtual void transform (const std::vector< GPType > &location, std::vector< GPType > &out) const =0

  Transforms the specified location using the properties (e.g.
- virtual bool isPeriodic () const =0

  Gets whether or not this grid is periodic (i.e.
- virtual ProjectionInfoPacket ∗ **getProjectionInfo** (AgentId id, bool secondaryInfo=false, std::set< AgentId > ∗secondaryIds=0, int destProc=-1)=0
- virtual void **updateProjectionInfo** (ProjectionInfoPacket ∗pip, Context< T > ∗context)=0
- virtual void getRequiredAgents (std::set< AgentId > &agentsToTest, std::set< AgentId > &agentsRequired, RADIUS radius=Projection< T >::PRIMARY)

  Given a set of agents to test, gets the subset that must be kept in order to fulfill the projection's 'contract' to the specified radius.
- virtual void getAgentsToPush (std::set< AgentId > &agentsToTest, std::map< int, std::set< AgentId > > &agentsToPush)=0

  Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.
- virtual bool keepsAgentsOnSyncProj ()

  Should return true if the Projection implemented can 'keep' some (non-local) agents during a projection information synchronization operation.
- virtual bool sendsSecondaryAgentsOnStatusExchange ()

  Should return true if the Projection implemented will send secondary agents during a status exchange.
- virtual void getInfoExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceiveFrom)=0

  Gets the set of processes with which this Projection exchanges projection info.
- virtual void getAgentStatusExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceiveFrom)=0

  Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.
- virtual void **cleanProjectionInfo** (std::set< AgentId > &agentsToKeep)

Additional Inherited Members

5.41.1 Detailed Description

template<typename T, typename GPType>class repast::Grid< T, GPType >

Abstract interface for Grids and ContinuousSpaces.

Template Parameters

T	the type of objects this Grid contains
GPType	the coordinate type of the grid point locations. This must be an int or a double.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 template<typename T, typename GPType> repast::Grid< T, GPType >::Grid (std::string *name*) [inline]

Creates a Grid with the specified name.

Parameters

name	the name of the Grid. This should be unique among Projections.

5.41.3 Member Function Documentation

5.41.3.1 template<typename T, typename GPType> virtual bool repast::Grid< T, GPType >::contains (const AgentId & *id*) `[pure virtual]`

Gets whether or not this grid contains the agent with the specified id.

Parameters

id	the id of the agent to check

Returns

true if the grid contains the agent, otherwise false.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GP-Transformer, Adder, int >.

5.41.3.2 template<typename T, typename GPType> virtual const GridDimensions repast::Grid< T, GPType >::dimensions ( ) const `[pure virtual]`

Gets the dimensions of this Grid.

Returns

the dimensions of this Grid.

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, G-PTransformer, Adder, int >, repast::SharedBaseGrid< T, GPTransformer, Adder, double >, repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GP-Transformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.3 template<typename T, typename GPType> virtual void repast::Grid< T, GPType >::getAgentStatus-ExchangePartners (std::set< int > & *psToSendTo*, std::set< int > & *psToReceiveFrom*) `[pure virtual]`

Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.

In the most general case this will be all other processors. However, simulations where agents move in spaces will usually exchange agents only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implements repast::Projection< T >.

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GP-Transformer, Adder, int >, and repast::SharedBaseGrid< T, GPTransformer, Adder, double >.

5.41.3.4 template<typename T, typename GPType> virtual void **repast::Grid**< T, GPType >::getAgentsToPush (std::set< **AgentId** > & *agentsToTest,* std::map< int, std::set< **AgentId** > > & *agentsToPush*) [pure virtual]

Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.

Generally spaces must push agents that are in 'buffer zones' and graphs must push local agents that are vertices to master edges where the other vertex is non- local. The results are returned per-process in the agentsToPush map.

Implements repast::Projection< T >.

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, repast::SharedBaseGrid< T, GPTransformer, Adder, double >, repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >, and repast::SharedDiscreteSpace< T, GPTransformer, Adder >.

5.41.3.5 template<typename T, typename GPType> virtual void **repast::Grid**< T, GPType >::getDisplacement (const **Point**< GPType > & *pt1,* const **Point**< GPType > & *pt2,* std::vector< GPType > & *out*) const [pure virtual]

Gets vector difference between point 1 and point 2, putting the result in out.

Parameters

	p1	the first point
	p2	the second point
out	*the*	vector where the difference will be put

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.6 template<typename T, typename GPType> virtual double **repast::Grid**< T, GPType >::getDistance (const **Point**< GPType > & *pt1,* const **Point**< GPType > & *pt2*) const [pure virtual]

Gets the distance between the two grid points.

Parameters

p1	the first point
p2	the second point

Returns

the distance between pt1 and pt2.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.7 template<typename T, typename GPType> virtual double **repast::Grid**< T, GPType >::getDistanceSq (const **Point**< GPType > & *pt1,* const **Point**< GPType > & *pt2*) const [pure virtual]

Gets the square of the distance between the two grid points.

Parameters

p1	the first point
p2	the second point

Returns

the square of the distance between pt1 and pt2.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.8 template<typename T, typename GPType> virtual void **repast::Grid**< T, GPType >::getInfoExchangePartners (std::set< int > & *psToSendTo,* std::set< int > & *psToReceiveFrom*) [pure virtual]

Gets the set of processes with which this Projection exchanges projection info.

In the most general case this will be all other processors; this is the case for graphs, where agent connections can be arbitrary. However, spaces usually exchange information only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implements repast::Projection< T >.

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, and repast::SharedBaseGrid< T, GPTransformer, Adder, double >.

5.41.3.9 template<typename T, typename GPType> virtual bool **repast::Grid**< T, GPType >::getLocation (const T * *agent,* std::vector< GPType > & *out*) const [pure virtual]

Gets the location of this agent and puts it in the specified vector.

The x coordinate will be the first value, the y the second and so on.

Parameters

	agent	the agent whose location we want to get
out	*the*	vector where the agents location will be put

Returns

true if the location was successfully found, otherwise false.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.10 template<typename T, typename GPType> virtual bool **repast::Grid**< T, GPType >::getLocation (const **AgentId** & *id,* std::vector< GPType > & *out*) const [pure virtual]

Gets the location of this agent and puts it in the specified vectors.

The x coordinate will be the first value, the y the second and so on.

Parameters

	id	the id of the agent whose location we want to get
out	*out*	the agent's location will be put into this vector

Returns

true if the location was successfully found, otherwise false.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.11 template<typename T, typename GPType> virtual T∗ repast::Grid< T, GPType >::getObjectAt (const Point< GPType > & *pt*) const [pure virtual]

Gets the first object found at the specified point, or NULL if there is no such object.

Returns

the first object found at the specified point, or NULL if there is no such object.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.12 template<typename T, typename GPType> virtual void repast::Grid< T, GPType >::getObjectsAt (const Point< GPType > & *pt*, std::vector< T ∗ > & *out*) const [pure virtual]

Gets all the objects found at the specified point.

The found objects will be put into the out parameter.

Parameters

	pt	the point to get all the objects at
out	*out*	the vector into which the found objects will be put

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.13 template<typename T, typename GPType> virtual void repast::Grid< T, GPType >::getRequiredAgents (std::set< AgentId > & *agentsToTest*, std::set< AgentId > & *agentsRequired*, RADIUS *radius* = Projection< T >::PRIMARY) [inline], [virtual]

Given a set of agents to test, gets the subset that must be kept in order to fulfill the projection's 'contract' to the specified radius.

Generally spaces do not require any agents, but graphs do- generally the non-local ends to master copies of edges.

Implements repast::Projection< T >.

5.41.3.14 template<typename T, typename GPType> virtual bool repast::Grid< T, GPType >::isPeriodic ( ) const [pure virtual]

Gets whether or not this grid is periodic (i.e.

toroidal).

Returns

true if this Grid is periodic, otherwise false.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GP-Transformer, Adder, int >.

5.41.3.15 template<typename T, typename GPType> virtual bool repast::Grid< T, GPType >::keepsAgentsOnSyncProj ( ) [inline], [virtual]

Should return true if the Projection implemented can 'keep' some (non-local) agents during a projection information synchronization operation.

Generally spaces will allow all non-local agents to be deleted, but graphs keep the non-local agents that participate in Master edges.

It is possible to override these. A graph projection can be created that does not permit non-local agents to be 'kept'. This would be an extremely unusual use case, but it is possible.

Note that these are used for optimization. If no projection in a given context keeps any agents, several steps in the synchronization algorithm can be omitted. Of course, omitting these steps when a projection actually retains agents can caused undefined problems.

Returns

true if this projection will keep non-local agents during a projection information synchronziation event, false if it will not.

Implements repast::Projection< T >.

5.41.3.16 template<typename T, typename GPType> virtual std::pair<bool, Point<GPType> > repast::Grid< T, GPType >::moveByDisplacement (const T ∗ *agent*, const std::vector< GPType > & *displacement*) [pure virtual]

Moves the specified object from its current location by the specified amount.

For example `moveByDisplacement(object, 3, -2, 1)` will move the object by 3 along the x-axis, -2 along the y and 1 along the z. The displacement argument can be less than the number of dimensions in the space in which case the remaining argument will be set to 0. For example, `moveByDisplacement(object, 3)` will move the object 3 along the x-axis and 0 along the y and z axes, assuming a 3D grid.

Parameters

agent	the object to move
displacement	the amount to move the object

Returns

a pair containing a bool that indicates whether the move was a success or not, and the point where the agent was moved to.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GP-Transformer, Adder, int >.

5.41.3.17 template< typename T, typename GPType > virtual std::pair< bool, Point< GPType > > repast::Grid< T, GPType >::moveByVector (const T * *agent*, double *distance*, const std::vector< double > & *anglesInRadians*) [pure virtual]

Moves the specifed object the specified distance from its current position along the specified angle.

For example, `moveByVector(object, 1, Grid.NORTH)` will move the object 1 unit "north" up the y-axis, assuming a 2D grid. Similarly, `grid.moveByVector(object, 2, 0, Math.toRadians(90), 0)` will rotate 90 degrees around the y-axis, thus moving the object 2 units along the z-axis.

Note that the radians / degrees are incremented in a anti-clockwise fashion, such that 0 degrees is "east", 90 degrees is "north", 180 is "west" and 270 is "south."

Parameters

agent	**the object to move**
distance	**the distance to move**
anglesIn-Radians	**the angle to move along in radians.**

Returns

a pair containing a bool that indicates whether the move was a success or not, and the point where the agent was moved to.

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GP-Transformer, Adder, int >.

5.41.3.18 template< typename T, typename GPType > virtual bool repast::Grid< T, GPType >::moveTo (const AgentId & *id*, const Point< GPType > & *pt*) [pure virtual]

Moves the specified agent to the specified point.

Parameters

id	the id of the agent to move
pt	where to move the agent to

Returns

true if the move was successful, otherwise false

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, G-PTransformer, Adder, int >, repast::SharedBaseGrid< T, GPTransformer, Adder, double >, repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GP-Transformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >.

5.41.3.19 template< typename T, typename GPType > virtual bool repast::Grid< T, GPType >::sendsSecondaryAgentsOnStatusExchange ( ) [inline], [virtual]

Should return true if the Projection implemented will send secondary agents during a status exchange.

Generally spaces do not and graphs do.

If no secondary agents will be sent, portions of the algorithm can be omitted for optimization.

Returns

true if the Projection returns secondary agents, false if not

Implements repast::Projection< T >.

5.41.3.20 template<typename T, typename GPType> virtual void **repast::Grid**< T, GPType >::transform (const std::vector< GPType > & *location*, std::vector< GPType > & *out*) const [pure virtual]

Transforms the specified location using the properties (e.g.

toroidal) of this space.

Parameters

	location	the location to transform
out	*out*	the vector where the result of the transform will be put

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GP-Transformer, Adder, int >.

5.41.3.21 template<typename T, typename GPType> virtual void **repast::Grid**< T, GPType >::translate (const **Point**< GPType > & *location*, const **Point**< GPType > & *displacement*, std::vector< GPType > & *out*) const [pure virtual]

Translates the specified location by the specified displacement put the result in out.

Parameters

	location	the initial location
	displacement	the amount to translate the location by
out	*out*	the vector where the result of the translation is put

Implemented in repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, and repast::BaseGrid< T, MultipleOccupancy< T, int >, GP-Transformer, Adder, int >.

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

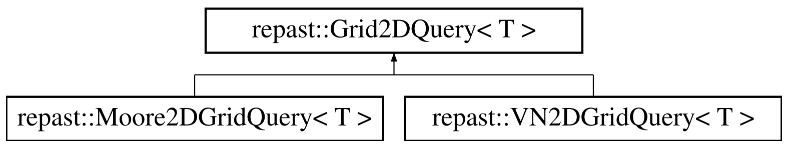
- repast_hpc/Grid.h

5.42 repast::Grid2DQuery< T > Class Template Reference

Base class for neighborhood queries on discrete Grids.

`#include <Grid2DQuery.h>`

Inheritance diagram for repast::Grid2DQuery< T >:



Public Member Functions

- Grid2DQuery (const Grid< T, int > ∗grid)

  Creates Grid2DQuery that will query the specified Grid.
- virtual void query (const Point< int > &center, int range, bool includeCenter, std::vector< T ∗ > &out) const =0

  Queries the Grid for the neighbors surrounding the center point within a specified range.

Protected Attributes

- const Grid< T, int > ∗ **_grid**
- int **minMax** [2][2]

5.42.1 Detailed Description

template<typename T>class repast::Grid2DQuery< T >

Base class for neighborhood queries on discrete Grids.

Template Parameters

T	the type of object in the Grid.

5.42.2 Member Function Documentation

5.42.2.1 template<typename T > virtual void repast::Grid2DQuery< T >::query (const Point< int > & *center,* int *range,* bool *includeCenter,* std::vector< T ∗ > & *out*) const `[pure virtual]`

Queries the Grid for the neighbors surrounding the center point within a specified range.

What constitutes the neighborhood is determines by subclass implementors.

Parameters

	center	the center of the neighborhood
	range	the range of the neighborhood out from the center
	includeCenter	whether or not to include any agents at the center
`out`	*the*	neighboring agents will be returned in this vector

Implemented in repast::Moore2DGridQuery< T >, and repast::VN2DGridQuery< T >.

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

- repast_hpc/Grid2DQuery.h

5.43 repast::GridBufferSyncher< T, GPType > Class Template Reference

DEPRECATED Helper class that provides support for synchronizing a grid / space buffer.

`#include <SharedBaseGrid.h>`

Public Member Functions

- **GridBufferSyncher** (boost::mpi::communicator ∗world)
- std::vector< CellContents< T,
  GPType > > ∗ **received** (size_t index)

- int **nghRank** (size_t index)
- size_t **vecsSize** ()
- void send (int rank, std::vector< CellContents< T, GPType > > &contents, int tag)

  Sends the contents to the rank.
- void **receive** (Neighbor ∗ngh, int tag)
- void **wait** ()

5.43.1 Detailed Description

template<typename T, typename GPType>class repast::GridBufferSyncher< T, GPType >

DEPRECATED Helper class that provides support for synchronizing a grid / space buffer.

Deprecated As of Version 2.0

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

- repast_hpc/SharedBaseGrid.h

5.44 repast::GridDimensions Class Reference

Basic structure for specifying grid dimenions.

```
#include <GridDimensions.h>
```

Public Member Functions

- **GridDimensions** (Point< double > extent)
- GridDimensions (Point< double > origin, Point< double > extent)

  Creates a GridDimensions with the specified origin and extent.
- bool **contains** (const Point< int > &pt) const
- bool **contains** (const std::vector< int > &pt) const
- bool **contains** (const Point< double > &pt) const
- bool **contains** (const std::vector< double > &pt) const
- const Point< double > & origin () const

  Gets the origin.
- const Point< double > & extents () const

  Gets the extents along each dimension.
- const double & **origin** (int index) const
- const double & **extents** (int index) const
- size_t **dimensionCount** () const

Friends

- bool **operator==** (const GridDimensions &one, const GridDimensions &two)
- std::ostream & **operator<<** (std::ostream &os, const GridDimensions &dimensions)

5.44.1 Detailed Description

Basic structure for specifying grid dimenions.

Structure is to specify (using instances of Point) the origin and the extent, so that an origin of (-100, -100) and an extent of (200, 200) represents a rectangle with corners at (-100, -100), (-100, 100), (100, 100), and (100, -100).

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

- repast_hpc/GridDimensions.h
- repast_hpc/GridDimensions.cpp

5.45 repast::GridMovePacket< PtType > Struct Template Reference

Encapsulates info about an agent moving off the grid: the rank it moved to, its grid location, and the agent id.

```
#include <GridMovePackets.h>
```

Public Member Functions

- **GridMovePacket** (const Point< PtType > &pt, const AgentId &id, int rank)
- template< class Archive >
  void **serialize** (Archive &ar, const unsigned int version)

Public Attributes

- Point< PtType > **_pt**
- AgentId **_id**
- int **_rank**

5.45.1 Detailed Description

template<typename PtType>struct repast::GridMovePacket< PtType >

Encapsulates info about an agent moving off the grid: the rank it moved to, its grid location, and the agent id.

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

- repast_hpc/GridMovePackets.h

5.46 repast::GridMovePackets< PtType > Class Template Reference

A collection of GridMovePacket objects, kept in a map per destination process.

```
#include <GridMovePackets.h>
```

Public Member Functions

- void **addPacket** (const GridMovePacket< PtType > &packet)
- void **clear** ()
- void removePacketFor (const AgentId &id)

  Removes any GridMovePacket-s associated with the specified agent id.
- void **send** (std::vector< boost::mpi::request > &requests, boost::mpi::communicator world)
- void **receivers** (std::vector< int > &receivers)

5.46.1 Detailed Description

template<typename PtType>class repast::GridMovePackets< PtType >

A collection of GridMovePacket objects, kept in a map per destination process.

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

- repast_hpc/GridMovePackets.h

5.47 repast::GridPointHolder< T, GPType > Struct Template Reference

Encapsulates a grid point and what is held in it.

`#include <BaseGrid.h>`

Public Attributes

- bool **inGrid**
- Point< GPType > **point**
- boost::shared_ptr< T > **ptr**

5.47.1 Detailed Description

template<typename T, typename GPType>struct repast::GridPointHolder< T, GPType >

Encapsulates a grid point and what is held in it.

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

- repast_hpc/BaseGrid.h

5.48 repast::HashGridPoint< T > Struct Template Reference

Class that allows retrieval of hash value for Point objects.

`#include <Point.h>`

Public Member Functions

- std::size_t **operator()** (const Point< T > &pt) const

5.48.1 Detailed Description

template<typename T>struct repast::HashGridPoint< T >

Class that allows retrieval of hash value for Point objects.

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

- repast_hpc/Point.h

5.49 repast::HashId Struct Reference

operator() implementation that returns the hashcode of an AgentId.

`#include <AgentId.h>`

Public Member Functions

- std::size_t **operator()** (const AgentId &id) const

5.49.1 Detailed Description

operator() implementation that returns the hashcode of an AgentId.

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

- repast_hpc/AgentId.h

5.50 repast::HashVertex< V, E > Struct Template Reference

Hashes a Vertex using the hashcode of the AgentId that the vertex contains.

`#include <Vertex.h>`

Public Member Functions

- std::size_t **operator()** (Vertex< V, E > ∗vertex) const

5.50.1 Detailed Description

template<typename V, typename E>struct repast::HashVertex< V, E >

Hashes a Vertex using the hashcode of the AgentId that the vertex contains.

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

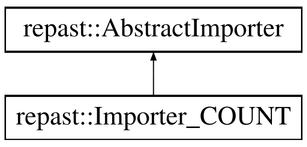
- repast_hpc/Vertex.h

5.51 repast::Importer_COUNT Class Reference

Importer that maintains a simple count of the agents being sent from each sending process.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::Importer_COUNT:



Public Member Functions

- virtual void registerOutgoingRequests (AgentRequest &req)

  Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.
- virtual void importedAgentIsRemoved (const AgentId &id)

  Notifies this importer that the agent that it (presumably) has been importing has been removed from the simulation on its home process, and the information for that agent will no longer be sent.
- virtual void importedAgentIsMoved (const AgentId &id, int newProcess)

  Notifies this importer that the agent that it (presumably) has been importing from another process has been moved; its information will now be received from its new home process (unless the agent was moved to this process)
- virtual std::string getReport ()

  Get a printable indication of the data in this object.
- virtual void **getSetOfAgentsBeingImported** (std::set< AgentId > &set)

Additional Inherited Members

5.51.1 Detailed Description

Importer that maintains a simple count of the agents being sent from each sending process.

When the count for a given process is zero no mpi 'receive' is created for that process; when the count is nonzero, a 'receive' is created. Note that no record of which agents are requested is kept, only the count of agents requested is kept.

5.51.2 Member Function Documentation

5.51.2.1 void Importer_COUNT::registerOutgoingRequests (**AgentRequest** & *req*) `[virtual]`

Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.

The record must at a minimum indicate which other processes are sending agent information, but may include other information, such as how many times a particular agent has been requested.

Implements repast::AbstractImporter.

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

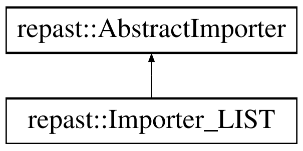
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.52 repast::Importer_LIST Class Reference

Importer that maintains a list of the agents being sent from each sending process.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::Importer_LIST:



Public Member Functions

- virtual void registerOutgoingRequests (AgentRequest &req)

  Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.
- virtual void importedAgentIsRemoved (const AgentId &id)

  Notifies this importer that the agent that it (presumably) has been importing has been removed from the simulation on its home process, and the information for that agent will no longer be sent.
- virtual void importedAgentIsMoved (const AgentId &id, int newProcess)

  Notifies this importer that the agent that it (presumably) has been importing from another process has been moved; its information will now be received from its new home process (unless the agent was moved to this process)
- virtual std::string getReport ()

  Get a printable indication of the data in this object.
- virtual void **getSetOfAgentsBeingImported** (std::set< AgentId > &set)
- virtual void **clear** ()

Additional Inherited Members

5.52.1 Detailed Description

Importer that maintains a list of the agents being sent from each sending process.

If there are no agents being sent the size of the list will be zero and no mpi 'receive' will be created; for non-zero-length lists a single mpi 'receiver' is created. An agent that is requested twice is placed in the list twice. Semantically this means that an agent cancellation will remove the agent from the list exactly once, but also that removing an agent from the list who is not found in the list will not reduce the list size (c.f. the 'count' version, where every cancellation reduces the count whether the specific agent being canceled was ever requested at all).

5.52.2 Member Function Documentation

5.52.2.1 void Importer_LIST::registerOutgoingRequests (**AgentRequest** & *req*) `[virtual]`

Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.

The record must at a minimum indicate which other processes are sending agent information, but may include other information, such as how many times a particular agent has been requested.

Implements repast::AbstractImporter.

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

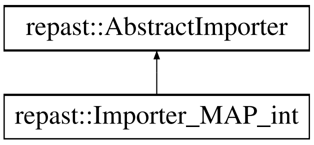
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.53 repast::Importer_MAP_int Class Reference

Importer that maintains a map of agents being sent from each sending process and a count of the number of times that agent was requested.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::Importer_MAP_int:



Public Member Functions

- virtual void registerOutgoingRequests (AgentRequest &req)

  Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.
- virtual void importedAgentIsRemoved (const AgentId &id)

  Notifies this importer that the agent that it (presumably) has been importing has been removed from the simulation on its home process, and the information for that agent will no longer be sent.
- virtual void importedAgentIsMoved (const AgentId &id, int newProcess)

  Notifies this importer that the agent that it (presumably) has been importing from another process has been moved; its information will now be received from its new home process (unless the agent was moved to this process)
- virtual std::string getReport ()

  Get a printable indication of the data in this object.
- virtual void **getSetOfAgentsBeingImported** (std::set< AgentId > &set)
- virtual void **clear** ()

Additional Inherited Members

5.53.1 Detailed Description

Importer that maintains a map of agents being sent from each sending process and a count of the number of times that agent was requested.

This is semantically equivalent to a 'LIST' type, but may be more or less appropriate in specific contexts based on performance. Duplicate requests for an agent increment the count associated with that agent; canceling a request reduces that count. If a given sending process has no agents being shared, no mpi 'receive' will be created, but if there are agents being shared, a 'receive' will be created.

5.53.2 Member Function Documentation

5.53.2.1 void Importer_MAP_int::registerOutgoingRequests (**AgentRequest** & *req*) [virtual]

Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.

The record must at a minimum indicate which other processes are sending agent information, but may include other information, such as how many times a particular agent has been requested.

Implements repast::AbstractImporter.

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

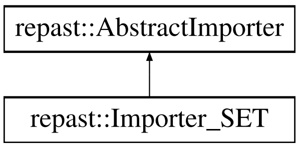
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.54 repast::Importer_SET Class Reference

Importer that maintains a set of agents being sent from each sending process.

```
#include <AgentImporterExporter.h>
```

Inheritance diagram for repast::Importer_SET:



Public Member Functions

- virtual void registerOutgoingRequests (AgentRequest &req)

  Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.
- virtual void importedAgentIsRemoved (const AgentId &id)

  Notifies this importer that the agent that it (presumably) has been importing has been removed from the simulation on its home process, and the information for that agent will no longer be sent.
- virtual void importedAgentIsMoved (const AgentId &id, int newProcess)

  Notifies this importer that the agent that it (presumably) has been importing from another process has been moved; its information will now be received from its new home process (unless the agent was moved to this process)
- virtual std::string getReport ()

  Get a printable indication of the data in this object.
- virtual void **getSetOfAgentsBeingImported** (std::set< AgentId > &set)
- virtual void **clear** ()

Additional Inherited Members

5.54.1 Detailed Description

Importer that maintains a set of agents being sent from each sending process.

Note that because this is a set, an agent may appear in it only once, no matter how many times it is requested. Canceling the agent will remove it from the set, even if the agent was requested multiple duplicate times. If the set for a given process is size zero, no mpi 'receive' will be created; if the set has any elements, an mpi 'receive' will be created.

5.54.2 Member Function Documentation

5.54.2.1 void Importer_SET::registerOutgoingRequests (**AgentRequest** & *req*) [virtual]

Given an agent request (including requests for agents on multiple other processes), makes a record of the agents that are being requested by this process and will therefore be received from other processes.

The record must at a minimum indicate which other processes are sending agent information, but may include other information, such as how many times a particular agent has been requested.

Implements repast::AbstractImporter.

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

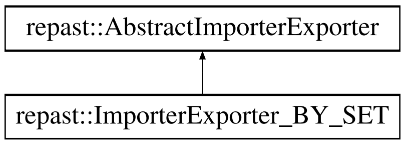
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.55 repast::ImporterExporter_BY_SET Class Reference

Implementation of the AbstractImporterExporter class that wraps a collection of AbstractImporterExporter objects that can be referenced by name.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::ImporterExporter_BY_SET:


repast::AbstractImporterExporter
repast::ImporterExporter_BY_SET

Public Member Functions

- virtual const std::set< int > & **getExportingProcesses** ()
- const std::set< int > & **getExportingProcesses** (std::string setName)
- virtual void **registerOutgoingRequests** (AgentRequest &request)
- void **registerOutgoingRequests** (AgentRequest &request, std::string setName, AGENT_IMPORTER_EXPORTER_TYPE setType=DEFAULT_ENUM_SYMBOL)
- virtual void **importedAgentIsRemoved** (const AgentId &id)
- virtual void **importedAgentIsMoved** (const AgentId &id, int newProcess)
- virtual void **importedAgentIsNowLocal** (const AgentId &id)
- virtual const AbstractExporter::StatusMap $*$ **getOutgoingStatusChanges** ()
- virtual const std::set< int > & **getProcessesExportedTo** ()
- const std::set< int > & **getProcessesExportedTo** (std::string setName)
- virtual void **registerIncomingRequests** (std::vector< AgentRequest > &requests)
- void **registerIncomingRequests** (std::vector< AgentRequest > &requests, std::string setName)
- virtual void **agentRemoved** (const AgentId &id)
- virtual void **agentMoved** (const AgentId &id, int newProcess)
- virtual void **incorporateAgentExporterInfo** (std::map< int, AgentRequest $*$ > info)
- void **incorporateAgentExporterInfo** (std::map< std::string, std::map< int, AgentRequest $*$ > $*$ > info)
- virtual void **clearStatusMap** ()
- virtual AgentExporterInfo $*$ **getAgentExportInfo** (int destProc)
- virtual void **clearAgentExportInfo** ()
- virtual const std::map< int, AgentRequest > & **getAgentsToExport** ()
- const std::map< int, AgentRequest > & **getAgentsToExport** (std::string setName)
- virtual std::string version ()

  Returns the version of this AbstractImporterExporter.
- void **dropSet** (std::string setName)
- virtual std::string getReport ()

  Gets a printable report of the state of this object.
- virtual void **getSetOfAgentsBeingImported** (std::set< AgentId > &set)
- void **getSetOfAgentsBeingImported** (std::set< AgentId > &set, std::string excludeSet)
- virtual void **clear** ()
- void **clear** (std::string setName)
- virtual void **clearExporter** ()
- void **clearExporter** (std::string setName)
- void **clearExportToSpecificProc** (int rank)

Additional Inherited Members

5.55.1 Detailed Description

Implementation of the AbstractImporterExporter class that wraps a collection of AbstractImporterExporter objects that can be referenced by name.

Each object can contain a different collection of agents, which were requested with an associated name for the importer/exporter to be used. They can then be updated separately, allowing for improved performance under certain circumstances.

5.55.2 Member Function Documentation

5.55.2.1 std::string ImporterExporter_BY_SET::version ( ) [virtual]

Returns the version of this AbstractImporterExporter.

The version is a string that indicates the semantic version of the importer and the exporter (e.g. "COUNT_LIST")

Implements repast::AbstractImporterExporter.

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

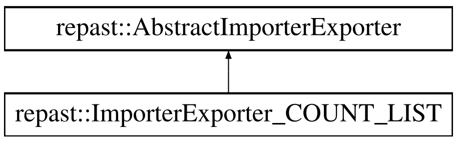
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.56 repast::ImporterExporter_COUNT_LIST Class Reference

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_COUNT' and an exporter of type 'Exporter_LIST'.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::ImporterExporter_COUNT_LIST:


repast::AbstractImporterExporter
repast::ImporterExporter_COUNT_LIST

Public Member Functions

- **ImporterExporter_COUNT_LIST** (AbstractExporter::StatusMap ∗outgoingStatusMap, AgentExporterData ∗outgoingAgentExporterInfo)
- virtual std::string version ()

  Returns the version of this AbstractImporterExporter.

Additional Inherited Members

5.56.1 Detailed Description

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_COUNT' and an exporter of type 'Exporter_LIST'.

5.56.2 Member Function Documentation

5.56.2.1 std::string ImporterExporter_COUNT_LIST::version ( ) [virtual]

Returns the version of this AbstractImporterExporter.

The version is a string that indicates the semantic version of the importer and the exporter (e.g. "COUNT_LIST")

Implements repast::AbstractImporterExporter.

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

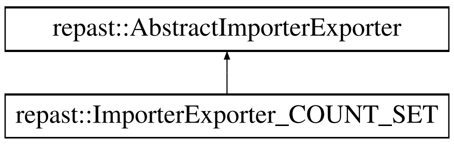
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.57 repast::ImporterExporter_COUNT_SET Class Reference

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_COUNT' and an exporter of type 'Exporter_SET'.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::ImporterExporter_COUNT_SET:


repast::AbstractImporterExporter
repast::ImporterExporter_COUNT_SET

Public Member Functions

- **ImporterExporter_COUNT_SET** (AbstractExporter::StatusMap ∗outgoingStatusMap, AgentExporterData ∗outgoingAgentExporterInfo)
- virtual std::string version ()

  Returns the version of this AbstractImporterExporter.

Additional Inherited Members

5.57.1 Detailed Description

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_COUNT' and an exporter of type 'Exporter_SET'.

5.57.2 Member Function Documentation

5.57.2.1 std::string ImporterExporter_COUNT_SET::version ( ) [virtual]

Returns the version of this AbstractImporterExporter.

The version is a string that indicates the semantic version of the importer and the exporter (e.g. "COUNT_LIST")

Implements repast::AbstractImporterExporter.

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

- repast_hpc/AgentImporterExporter.h

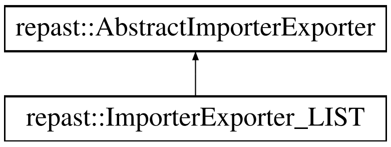
- repast_hpc/AgentImporterExporter.cpp

5.58 repast::ImporterExporter_LIST Class Reference

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_LIST' and an exporter of type 'Exporter_LIST'.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::ImporterExporter_LIST:


repast::AbstractImporterExporter
repast::ImporterExporter_LIST

Public Member Functions

- **ImporterExporter_LIST** (AbstractExporter::StatusMap ∗outgoingStatusMap, AgentExporterData ∗outgoingAgentExporterInfo)
- virtual std::string version ()

  Returns the version of this AbstractImporterExporter.

Additional Inherited Members

5.58.1 Detailed Description

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_LIST' and an exporter of type 'Exporter_LIST'.

5.58.2 Member Function Documentation

5.58.2.1 std::string ImporterExporter_LIST::version ( ) `[virtual]`

Returns the version of this AbstractImporterExporter.

The version is a string that indicates the semantic version of the importer and the exporter (e.g. "COUNT_LIST")

Implements repast::AbstractImporterExporter.

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

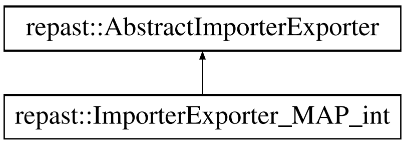
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.59 repast::ImporterExporter_MAP_int Class Reference

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_MAP_int' and an exporter of type 'Exporter_LIST'.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::ImporterExporter_MAP_int:


repast::AbstractImporterExporter
repast::ImporterExporter_MAP_int

Public Member Functions

- **ImporterExporter_MAP_int** (AbstractExporter::StatusMap ∗outgoingStatusMap, AgentExporterData ∗outgoingAgentExporterInfo)
- virtual std::string version ()

    Returns the version of this AbstractImporterExporter.

Additional Inherited Members

5.59.1 Detailed Description

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_MAP_int' and an exporter of type 'Exporter_LIST'.

5.59.2 Member Function Documentation

5.59.2.1 std::string ImporterExporter_MAP_int::version ( ) [virtual]

Returns the version of this AbstractImporterExporter.

The version is a string that indicates the semantic version of the importer and the exporter (e.g. "COUNT_LIST")

Implements repast::AbstractImporterExporter.

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

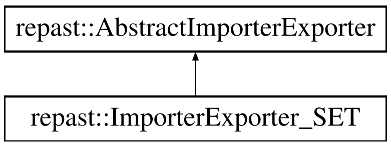
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.60 repast::ImporterExporter_SET Class Reference

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_SET' and an exporter of type 'Exporter_LIST'.

`#include <AgentImporterExporter.h>`

Inheritance diagram for repast::ImporterExporter_SET:


repast::AbstractImporterExporter
repast::ImporterExporter_SET

Public Member Functions

- **ImporterExporter_SET** (AbstractExporter::StatusMap ∗outgoingStatusMap, AgentExporterData ∗outgoingAgentExporterInfo)

- virtual std::string version ()

  Returns the version of this AbstractImporterExporter.

Additional Inherited Members

5.60.1 Detailed Description

An implementation of AbstractImporterExporter that uses an importer of type 'Importer_SET' and an exporter of type 'Exporter_LIST'.

5.60.2 Member Function Documentation

5.60.2.1 std::string ImporterExporter_SET::version ( ) `[virtual]`

Returns the version of this AbstractImporterExporter.

The version is a string that indicates the semantic version of the importer and the exporter (e.g. "COUNT_LIST")

Implements repast::AbstractImporterExporter.

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

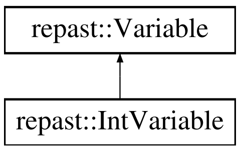
- repast_hpc/AgentImporterExporter.h
- repast_hpc/AgentImporterExporter.cpp

5.61 repast::IntVariable Class Reference

Used in SVDataSet to manage integer data.

`#include <Variable.h>`

Inheritance diagram for repast::IntVariable:



Public Member Functions

- virtual void write (size_t index, std::ofstream &out)

  Writes the data at the specified index to the specified ofstream.
- virtual void insert (double ∗array, size_t size)

  Inserts all the doubles in the double array into the collection of data stored in this Variable.
- virtual void insert (int ∗array, size_t size)

  Inserts all the ints in the int array into the collection of data stored in this Variable.
- virtual void clear ()

  Clears this Variable of all the data stored in it.

5.61.1 Detailed Description

Used in SVDataSet to manage integer data.

5.61.2 Member Function Documentation

5.61.2.1 void repast::IntVariable::insert (double ∗ *array*, size_t *size*) `[virtual]`

Inserts all the doubles in the double array into the collection of data stored in this Variable.

Parameters

array	the array to insert
size	the size of the array

Implements repast::Variable.

5.61.2.2 void repast::IntVariable::insert (int ∗ *array*, size_t *size*) `[virtual]`

Inserts all the ints in the int array into the collection of data stored in this Variable.

Parameters

array	the array to insert
size	the size of the array

Implements repast::Variable.

5.61.2.3 void repast::IntVariable::write (size_t *index*, std::ofstream & *out*) `[virtual]`

Writes the data at the specified index to the specified ofstream.

Parameters

index	the index of the data to write
out	the ofstream to write the data to

Implements repast::Variable.

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

- repast_hpc/Variable.h
- repast_hpc/Variable.cpp

5.62 repast::IsAgentType< T > Struct Template Reference

Struct that allows filtering by Agent Type.

`#include <AgentId.h>`

Public Member Functions

- **IsAgentType** (int typeId)
- bool **operator()** (const boost::shared_ptr< T > &ptr)
- bool **operator()** (const T ∗agent)

Public Attributes

- int **_typeId**

5.62.1 Detailed Description

template<typename T>struct repast::IsAgentType< T >

Struct that allows filtering by Agent Type.

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

- repast_hpc/AgentId.h

5.63 repast::IsLocalAgent< T > Struct Template Reference

Used in a filter iterator to filter on local agents only.

```
#include <SharedContext.h>
```

Public Member Functions

- **IsLocalAgent** (int rankInCommunicator)
- bool **operator()** (const boost::shared_ptr< T > &ptr)

Public Attributes

- int **rank**

5.63.1 Detailed Description

template<typename T>struct repast::IsLocalAgent< T >

Used in a filter iterator to filter on local agents only.

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

- repast_hpc/SharedContext.h

5.64 repast::IsNotType< T > Struct Template Reference

Struct that allows filtering by !(Agent Type)

```
#include <AgentId.h>
```

Public Member Functions

- **IsNotType** (int typeId)
- bool **operator()** (const boost::shared_ptr< T > &ptr)
- bool **operator()** (const T $*$agent)

Public Attributes

- int **_typeId**

5.64.1 Detailed Description

template<typename T>struct repast::IsNotType< T >

Struct that allows filtering by !(Agent Type)

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

- repast_hpc/AgentId.h

5.65 repast::ItemReceipt< E > Class Template Reference

DEPRECATED Receipt for edges Class used to receive edges being sent.

`#include <SharedNetwork.h>`

Public Member Functions

- **ItemReceipt** (int source)

Public Attributes

- std::vector< E > **items**
- int **source_**

5.65.1 Detailed Description

template<typename E>class repast::ItemReceipt< E >

DEPRECATED Receipt for edges Class used to receive edges being sent.

Deprecated As of Version 2.0 replaced by ProjectionInfoPacket objects.

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

- repast_hpc/SharedNetwork.h

5.66 repast::KEBuilder< V, E, Ec, EcM > Class Template Reference

Buils KE type networks.

`#include <NetworkBuilder.h>`

Public Member Functions

- void build (repast::Properties &props, repast::Graph< V, E, Ec, EcM > *graph)

  Builds the network.

5.66.1 Detailed Description

template$<$typename V, typename E, typename Ec, typename EcM$>$class repast::KEBuilder$<$ V, E, Ec, EcM $>$

Buils KE type networks.

See Klemm and Eguiluz, "Growing scale-free network with small world behavior" in Phys. Rev. E 65.

5.66.2 Member Function Documentation

5.66.2.1 template$<$typename V , typename E , typename Ec , typename EcM $>$ void repast::KEBuilder$<$ V, E, Ec, EcM $>$::build (repast::Properties & *props*, repast::Graph$<$ V, E, Ec, EcM $>$ $*$ *graph*)

Builds the network.

The graph should contains the vertices the build the network with and props should contain the M values.

Parameters

props	a Properties containing a property "ke.model.m" that specifies the M value.
graph	the graph to build the network

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

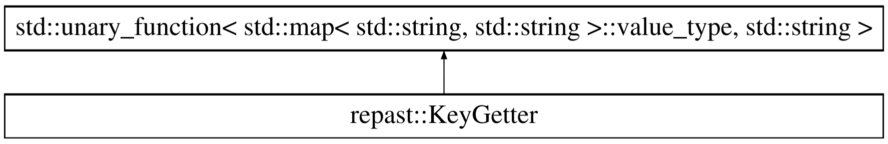
- repast_hpc/NetworkBuilder.h

5.67 repast::KeyGetter Struct Reference

Unary function used in a transform_iterator that allows the map iterator to return the keys.

`#include <Properties.h>`

Inheritance diagram for repast::KeyGetter:


std::unary_function< std::map< std::string, std::string >::value_type, std::string >
repast::KeyGetter

Public Member Functions

- std::string **operator()** (const std::map$<$ std::string, std::string $>$::value_type &value) const

5.67.1 Detailed Description

Unary function used in a transform_iterator that allows the map iterator to return the keys.

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

- repast_hpc/Properties.h
- repast_hpc/Properties.cpp

5.68 Log4CL Class Reference

Public Member Functions

- Logger & **get_logger** (std::string logger_name)
- void **close** ()

Static Public Member Functions

- static Log4CL ∗ **instance** ()
- static void **configure** (int, const std::string &, boost::mpi::communicator ∗comm=0, int maxConfigFileSize=M-AX_CONFIG_FILE_SIZE)
- static void **configure** (int)

Protected Member Functions

- **Log4CL** (int)

Friends

- class **Log4CLConfigurator**

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

- repast_hpc/logger.h
- repast_hpc/logger.cpp

5.69 Log4CLConfigurator Class Reference

Public Member Functions

- Log4CL ∗ **configure** (const std::string &config_file, int proc_id, boost::mpi::communicator ∗comm=0, int max-ConfigFileSize=MAX_CONFIG_FILE_SIZE)

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

- repast_hpc/logger.h
- repast_hpc/logger.cpp

5.70 Logger Class Reference

Public Member Functions

- **Logger** (const std::string, LOG_LEVEL, int proc_id)
- void **log** (LOG_LEVEL, const std::string msg)
- void **close** ()
- void **add_appender** (Appender ∗appender)

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

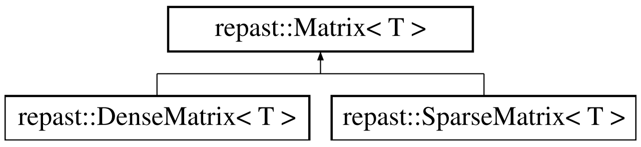
- repast_hpc/logger.h
- repast_hpc/logger.cpp

5.71 repast::Matrix< T > Class Template Reference

Base class for matrix implementations.

```
#include <matrix.h>
```

Inheritance diagram for repast::Matrix< T >:



Public Member Functions

- Matrix (const Point< int > &size, const T &defaultValue=T())

  Creates a matrix of the specified size and with the specified default value.
- virtual T & get (const Point< int > &index)=0

  Gets the value at the specified index.
- virtual void set (const T &value, const Point< int > &index)=0

  Sets the value at the specified index.
- T & **operator[]** (const Point< int > &index)
- const T & **operator[]** (const Point< int > &index) const
- const T & defaultValue () const

  Gets the default value of any unset matrix cell.
- const Point< int > shape () const

  Gets the shape (i.e.

Protected Member Functions

- int **calcIndex** (const Point< int > &index)
- void **boundsCheck** (const Point< int > &index)
- void **create** ()

Protected Attributes

- int * **stride**
- T **defValue**
- Point< int > **_size**
- int **dCount**

5.71.1 Detailed Description

template<typename T>class repast::Matrix< T >

Base class for matrix implementations.

5.71.2 Constructor & Destructor Documentation

5.71.2.1 template<typename T> repast::Matrix< T >::Matrix (const Point< int > & *size*, const T & *defaultValue* = `T()`) `[explicit]`

Creates a matrix of the specified size and with the specified default value.

Parameters

size	the size of the matrix in each dimension

5.71.3 Member Function Documentation

5.71.3.1 template<typename T> const Point<int> repast::Matrix< T >::shape ( ) const [inline]

Gets the shape (i.e.

the length of each dimensions) of the matrix.

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

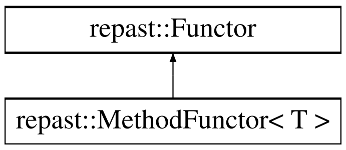
- repast_hpc/matrix.h

5.72 repast::MethodFunctor< T > Class Template Reference

Adapts a no-arg method call on an object instance to a Functor interface.

`#include <Schedule.h>`

Inheritance diagram for repast::MethodFunctor< T >:



Public Member Functions

- **MethodFunctor** (T ∗_obj, void(T::∗_fptr)())
- void **operator()** ()

5.72.1 Detailed Description

template<typename T>class repast::MethodFunctor< T >

Adapts a no-arg method call on an object instance to a Functor interface.

This is used by the Schedule code to schedule method calls on objects.

Template Parameters

T	the object type on which the call will be made.

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

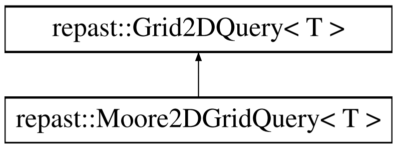
- repast_hpc/Schedule.h

5.73 repast::Moore2DGridQuery< T > Class Template Reference

Neighborhood query that gathers neighbors in a Moore (N, S, E, W, NE, etc.) neighborhood.

`#include <Moore2DGridQuery.h>`

Inheritance diagram for repast::Moore2DGridQuery< T >:



Public Member Functions

- Moore2DGridQuery (const Grid< T, int > ∗grid)

  Creates Moore2DGridQuery that will query the specified Grid.
- virtual void query (const Point< int > &center, int range, bool includeCenter, std::vector< T ∗ > &out) const

  Queries the Grid for the Moore neighbors surrounding the center point within a specified range.

Additional Inherited Members

5.73.1 Detailed Description

template<typename T>class repast::Moore2DGridQuery< T >

Neighborhood query that gathers neighbors in a Moore (N, S, E, W, NE, etc.) neighborhood.

Template Parameters

T	the type of agents in the Grid

5.73.2 Member Function Documentation

5.73.2.1 template<typename T > void **repast::Moore2DGridQuery**< T >::query (const **Point**< int > & *center,* int *range,* bool *includeCenter,* std::vector< T ∗ > & *out*) const [virtual]

Queries the Grid for the Moore neighbors surrounding the center point within a specified range.

Parameters

	center	the center of the neighborhood
	range	the range of the neighborhood out from the center
	includeCenter	whether or not to include any agents at the center
out	*the*	neighboring agents will be returned in this vector

Implements repast::Grid2DQuery< T >.

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

- repast_hpc/Moore2DGridQuery.h

5.74 repast::MultipleOccupancy< T, GPType > Class Template Reference

Multiple Occupancy cell accessor for accessing the occupants of locations in a Grid.

```
#include <MultipleOccupancy.h>
```

Public Member Functions

- T ∗ get (const Point< GPType > &location) const

    Gets the first object found at the specified location.
- void getAll (const Point< GPType > &location, std::vector< T ∗ > &out) const

    Gets all the items found at the specified location.
- bool put (boost::shared_ptr< T > &agent, const Point< GPType > &location)

    Puts the specified item at the specified location.
- void remove (boost::shared_ptr< T > &agent, const Point< GPType > &location)

    Removes the specified item from the specified location.

5.74.1 Detailed Description

template<typename T, typename GPType>class repast::MultipleOccupancy< T, GPType >

Multiple Occupancy cell accessor for accessing the occupants of locations in a Grid.

Each locations can have multiple occupants.

Parameters

T	the type of object in the Grid
GPType	the coordinate type of the grid point locations. This must be an int or a double.

5.74.2 Member Function Documentation

5.74.2.1 template<typename T , typename GPType> T ∗ repast::MultipleOccupancy< T, GPType >::get (const Point< GPType > & *location*) const

Gets the first object found at the specified location.

Parameters

location	the location to get the object at

Returns

the first object found at the specified location or 0 if there are no objects at the specified location.

5.74.2.2 template<typename T, typename GPType> void repast::MultipleOccupancy< T, GPType >::getAll (const Point< GPType > & *location*, std::vector< T ∗ > & *out*) const

Gets all the items found at the specified location.

Parameters

	location	the location to get the items at
`out`	*the*	found items will be returned in this vector

5.74.2.3 template<typename T, typename GPType> bool repast::MultipleOccupancy< T, GPType >::put (boost::shared_ptr< T > & *agent*, const Point< GPType > & *location*)

Puts the specified item at the specified location.

Parameters

agent	the item to put
location	the location to put the item at

5.74.2.4 template<typename T, typename GPType> void **repast::MultipleOccupancy**< T, GPType >::remove (boost::shared_ptr< T > & *agent,* const **Point**< GPType > & *location*)

Removes the specified item from the specified location.

Parameters

agent	the item to remove
location	the location to remove the item from

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

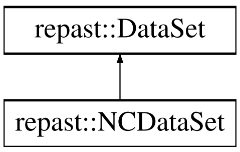
- repast_hpc/MultipleOccupancy.h

5.75 repast::NCDataSet Class Reference

Provides data recording and writing into a single file in NetCDF format.

```
#include <NCDataSet.h>
```

Inheritance diagram for repast::NCDataSet:



Public Member Functions

- void record ()
  Records the data.
- void write ()
  Writes the data.
- void close ()
  Closes the dataset, after which it must be recreated to be used.

Friends

- class **NCDataSetBuilder**

5.75.1 Detailed Description

Provides data recording and writing into a single file in NetCDF format.

A NCDataSet uses rank 0 to write to a single file from multiple pan-process data sources. A NCDataSet should be built using a NCDataSetBuilder.

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

- repast_hpc/NCDataSet.h
- repast_hpc/NCDataSet.cpp

5.76 repast::NCDataSetBuilder Class Reference

Used to build NCDataSets to record data in NetCDF format.

`#include <NCDataSetBuilder.h>`

Public Member Functions

- NCDataSetBuilder (std::string file, const Schedule &schedule)

  Creates an NCDataSetBuilder that will write to the specified file and get its tick counts from the specified schedule.
- NCDataSetBuilder & addDataSource (NCDataSource ∗source)

  Adds a NCDataSource to this NCDataSetBuilder.
- NCDataSet ∗ createDataSet ()

  Creates the NCDataSet defined by this NCDataSetBuilder.

5.76.1 Detailed Description

Used to build NCDataSets to record data in NetCDF format.

Steps for use are:

1. Create a NCDataSetBuilder.
2. Add NCDataSources to the builder using the createNCDataSource functions. Each DataSource defines a variable and where the data for that variable will be retrieved. Recording data on the NCDataSet produced by the builder will record this data for each variable.
3. Call createDataSet to create the NCDataSet.
4. Schedule calls to record and write on the NCDataSet.

5.76.2 Constructor & Destructor Documentation

5.76.2.1 repast::NCDataSetBuilder::NCDataSetBuilder (std::string *file,* const Schedule & *schedule*)

Creates an NCDataSetBuilder that will write to the specified file and get its tick counts from the specified schedule.

Parameters

file	the name of the file to write to. Only rank 0 will actually write to this file.
schedule	the schedule to get tick counts from

5.76.3 Member Function Documentation

5.76.3.1 NCDataSetBuilder & repast::NCDataSetBuilder::addDataSource (NCDataSource ∗ *source*)

Adds a NCDataSource to this NCDataSetBuilder.

The added NCDataSource defines a variable and where the data for that variable will be retrieved. Recording data on the NCDataSet produced by this builder will record this data for each variable.

5.76.3.2 NCDataSet ∗ repast::NCDataSetBuilder::createDataSet ( )

Creates the NCDataSet defined by this NCDataSetBuilder.

The caller is responsible for properly deleting the returned pointer.

Returns

the created NCDataSet.

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

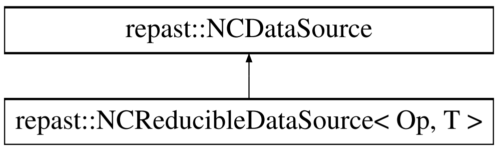
- repast_hpc/NCDataSetBuilder.h
- repast_hpc/NCDataSetBuilder.cpp

5.77 repast::NCDataSource Class Reference

Data source used internally by NCDataSets.

`#include <NCDataSource.h>`

Inheritance diagram for repast::NCDataSource:



Public Member Functions

- **NCDataSource** (std::string name)
- virtual void **record** ()=0
- virtual void **write** (NcVar ∗var)=0
- virtual NcType **ncType** ()=0
- const std::string **name** () const

Protected Attributes

- std::string **_name**

5.77.1 Detailed Description

Data source used internally by NCDataSets.

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

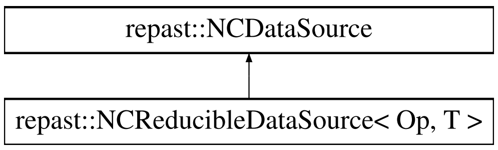
- repast_hpc/NCDataSource.h

5.78 repast::NCReducibleDataSource< Op, T > Class Template Reference

Source of data and a reduction operation.

`#include <NCReducibleDataSource.h>`

Inheritance diagram for repast::NCReducibleDataSource< Op, T >:



Public Member Functions

- **NCReducibleDataSource** (std::string name, TDataSource< T > ∗dataSource, Op op)
- virtual NcType **ncType** ()
- virtual void **record** ()
- virtual void **write** (NcVar ∗var)

Protected Attributes

- Op **op_**
- std::vector< T > **data**
- TDataSource< T > ∗ **dataSource_**
- int **rank**
- int **start**

5.78.1 Detailed Description

template<typename Op, typename T>class repast::NCReducibleDataSource< Op, T >

Source of data and a reduction operation.

Used internally by a NCDataSet to store the data sources. their associated ops etc.

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

- repast_hpc/NCReducibleDataSource.h

5.79 repast::NcTypeTrait< T > Struct Template Reference

Base class for specialized int and double NcType classes.

`#include <NCDataSource.h>`

5.79.1 Detailed Description

template<typename T>struct repast::NcTypeTrait< T >

Base class for specialized int and double NcType classes.

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

- repast_hpc/NCDataSource.h

5.80 repast::NcTypeTrait< double > Struct Template Reference

Used for converting to NetCDF Data, double type.

`#include <NCDataSource.h>`

Static Public Attributes

- static const NcType **type** = ncDouble

5.80.1 Detailed Description

template<>struct repast::NcTypeTrait< double >

Used for converting to NetCDF Data, double type.

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

- repast_hpc/NCDataSource.h

5.81 repast::NcTypeTrait< int > Struct Template Reference

Used for converting to NetCDF Data, int type.

`#include <NCDataSource.h>`

Static Public Attributes

- static const NcType **type** = ncInt

5.81.1 Detailed Description

template<>struct repast::NcTypeTrait< int >

Used for converting to NetCDF Data, int type.

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

- repast_hpc/NCDataSource.h

5.82 repast::Neighbor Class Reference

Contains the rank and boundaries of a semantically adjacent process (that is, a process that manages the space that is adjacent to the simulation space managed by this process).

`#include <SharedBaseGrid.h>`

Public Member Functions

- **Neighbor** (int rank, GridDimensions bounds)
- int **rank** () const
- GridDimensions **bounds** () const

5.82.1 Detailed Description

Contains the rank and boundaries of a semantically adjacent process (that is, a process that manages the space that is adjacent to the simulation space managed by this process).

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

- repast_hpc/SharedBaseGrid.h
- repast_hpc/SharedBaseGrid.cpp

5.83 repast::Neighbors Class Reference

Provides lookup of grid topology process neighbors given a point in the pan process grid.

`#include <SharedBaseGrid.h>`

Public Types

- enum Location {
  E, **W**, **N**, **S**,
  NE, **NW**, **SE**, **SW** }

  Describes the relative location of grid topology process neighbors.

Public Member Functions

- void addNeighbor (Neighbor ∗ngh, Neighbors::Location location)

  Adds a neighbor at the specified location.
- Neighbor ∗ neighbor (Neighbors::Location location) const

  Gets the neighbor at the specified location.
- Neighbor ∗ findNeighbor (const std::vector< int > &pt)

  Finds the neighbor that contains the specified point.
- Neighbor ∗ findNeighbor (const std::vector< double > &pt)

  Finds the neighbor that contains the specified point.
- void **getNeighborRanks** (std::set< int > &ranks)

Static Public Attributes

- static const int **LOCATION_SIZE** = 8

Friends

- std::ostream & **operator<<** (std::ostream &os, const Neighbors &nghs)

5.83.1 Detailed Description

Provides lookup of grid topology process neighbors given a point in the pan process grid.

5.83.2 Member Function Documentation

5.83.2.1 Neighbor ∗ repast::Neighbors::findNeighbor (const std::vector< int > & *pt*)

Finds the neighbor that contains the specified point.

Returns

the found neighbor

5.83.2.2 Neighbor ∗ repast::Neighbors::findNeighbor (const std::vector< double > & *pt*)

Finds the neighbor that contains the specified point.

Returns

the found neighbor

5.83.2.3 Neighbor ∗ repast::Neighbors::neighbor (Neighbors::Location *location*) const

Gets the neighbor at the specified location.

Parameters

location	the location of the neighbor.

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

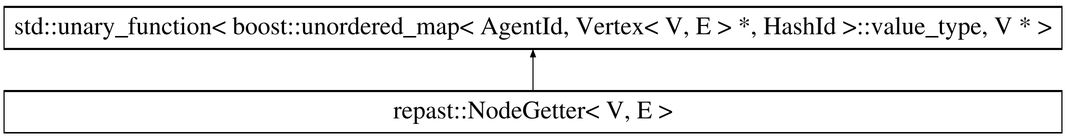
- repast_hpc/SharedBaseGrid.h
- repast_hpc/SharedBaseGrid.cpp

5.84 repast::NodeGetter< V, E > Struct Template Reference

Unary function used in the transform_iterator that allows an iterator over the vertex map to return the node.

```
#include <Vertex.h>
```

Inheritance diagram for repast::NodeGetter< V, E >:


std::unary_function< boost::unordered_map< AgentId, Vertex< V, E > *, HashId >::value_type, V * >
repast::NodeGetter< V, E >

Public Member Functions

- V ∗ **operator()** (const typename boost::unordered_map< AgentId, Vertex< V, E > ∗, HashId >::value_type &value) const

5.84.1 Detailed Description

template<typename V, typename E>struct repast::NodeGetter< V, E >

Unary function used in the transform_iterator that allows an iterator over the vertex map to return the node.

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

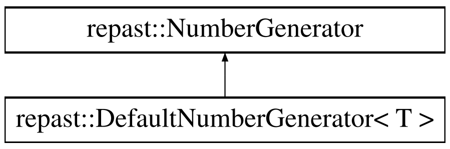
- repast_hpc/Vertex.h

5.85 repast::NumberGenerator Class Reference

Number generator interface.

`#include <Random.h>`

Inheritance diagram for repast::NumberGenerator:



Public Member Functions

- virtual double next ()=0

    Gets the "next" number from this Number Generator.

5.85.1 Detailed Description

Number generator interface.

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

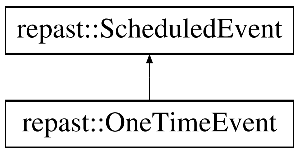
- repast_hpc/Random.h

5.86 repast::OneTimeEvent Class Reference

ScheduledEvent that will only execute only once.

`#include <Schedule.h>`

Inheritance diagram for repast::OneTimeEvent:



Public Member Functions

- **OneTimeEvent** (double, RepastEvent ∗)
- virtual bool reschedule (std::priority_queue< ScheduledEvent ∗, std::vector< ScheduledEvent ∗ >, Event-Compare > &)

    Always returns false, as it does not reschedule itself.

Additional Inherited Members

5.86.1 Detailed Description

ScheduledEvent that will only execute only once.

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

- repast_hpc/Schedule.h
- repast_hpc/Schedule.cpp

5.87 repast::Point< T > Class Template Reference

A N-dimensional Point representation.

`#include <Point.h>`

Public Types

- typedef std::vector< T >
  ::const_iterator **const_iterator**

Public Member Functions

- Point (T x)

  Creates a one dimensional point with the specified value.
- Point (T x, T y)

  Creates a two dimensional point with the specified values.
- Point (T x, T y, T z)

  Creates a three dimensional point with the specified values.
- Point (std::vector< T > coordinates)

  Creates a multi-dimensional point with the specified values.
- T getX () const

  Gets the x coordinate of the point.
- T getY () const

  Gets the y coordinate of the point.
- T getZ () const

  Gets the z coordinate of the point.
- T getCoordinate (int coordIndex) const

  Gets the coodinate of the point in the specified dimension.
- void add (const Point< T > &pt)

  Adds the specified GridPoint to this GridPoint.
- size_t dimensionCount () const

  Gets the number of dimensions of this point.
- const T & operator[] (size_t index) const

  Gets the coordinate value at the specified index.
- T & operator[] (size_t index)

  Gets the coordinate value at the specified index.
- const std::vector< T > & coords () const

  Gets the coordinates of this point as a vector.
- const_iterator begin () const

Gets the start of an iterator over the coordinates of this point.
- const_iterator end () const

  Gets the end of an iterator over the coordinates of this point.
- void copy (std::vector< T > &out) const

  Copies the point into the specified vector.

Friends

- struct **HashGridPoint**< **T** >
- class **boost::serialization::access**
- bool **operator==** (const Point< T > &one, const Point< T > &two)
- std::ostream & **operator**<< (std::ostream &os, const Point< T > &pt)

5.87.1 Detailed Description

template<typename T>class repast::Point< T >

A N-dimensional Point representation.

N dimensional point for addressing matrix locations.

Parameters

T	a numeric type. In repast and relogo these are limited to int and double.

5.87.2 Constructor & Destructor Documentation

5.87.2.1 template<typename T> repast::Point< T >::Point (T *x*) [explicit]

Creates a one dimensional point with the specified value.

Parameters

x	the x coordinate of the point

5.87.2.2 template<typename T> repast::Point< T >::Point (T *x*, T *y*)

Creates a two dimensional point with the specified values.

Parameters

x	the x coordinate of the point
y	the y coordinate of the point

5.87.2.3 template<typename T> repast::Point< T >::Point (T *x*, T *y*, T *z*)

Creates a three dimensional point with the specified values.

Parameters

x	the x coordinate of the point
y	the y coordinate of the point
z	the z coordinate of the point

5.87.2.4 template<typename T> repast::Point< T >::Point (std::vector< T > *coordinates*)

Creates a multi-dimensional point with the specified values.

Parameters

coordinates	the coordinate values of the point. The first element will be x, the second y and so on.

5.87.3 Member Function Documentation

5.87.3.1 template<typename T> void repast::Point< T >::add (const Point< T > & *pt*)

Adds the specified GridPoint to this GridPoint.

This GridPoint contains the result.

Exceptions

invalid_argument	exception if the pt doesn't have the same number of dimensions as this GridPoint.

5.87.3.2 template<typename T> const_iterator repast::Point< T >::begin ( ) const [inline]

Gets the start of an iterator over the coordinates of this point.

Returns

the start of an iterator over the coordinates of this point.

5.87.3.3 template<typename T> const std::vector<T>& repast::Point< T >::coords ( ) const [inline]

Gets the coordinates of this point as a vector.

Returns

a vector containing the coordinates of this point.

5.87.3.4 template<typename T> void repast::Point< T >::copy (std::vector< T > & *out*) const

Copies the point into the specified vector.

Assumes the array is the same length as this GridPoint.

Parameters

`out`	*the*	vector to copy the point coordinates into

5.87.3.5 template<typename T> size_t repast::Point< T >::dimensionCount ( ) const [inline]

Gets the number of dimensions of this point.

Returns

the number of dimensions of this point.

5.87.3.6 template<typename T> const_iterator repast::Point< T >::end ( ) const [inline]

Gets the end of an iterator over the coordinates of this point.

Returns

the end of an iterator over the coordinates of this point.

5.87.3.7 template<typename T > T repast::Point< T >::getCoordinate (int *coordIndex*) const

Gets the coodinate of the point in the specified dimension.

Parameters

coordIndex	the dimension of the point to get the coordinate for. X is the first, y is the second and so on.

Returns

the coordinate of the point in the specified dimension.

Exceptions

an	out_of_range exception if this GridPoint has doesn't have the specified dimension.

5.87.3.8 template<typename T > T repast::Point< T >::getX ( ) const

Gets the x coordinate of the point.

Returns

the x coordinate of the point.

5.87.3.9 template<typename T > T repast::Point< T >::getY ( ) const

Gets the y coordinate of the point.

Returns

the y coordinate of the point.

Exceptions

an	out_of_range exception if this GridPoint has less than 2 dimensions.

5.87.3.10 template<typename T > T repast::Point< T >::getZ ( ) const

Gets the z coordinate of the point.

Returns

the z coordinate of the point.

Exceptions

an	out_of_range exception if this GridPoint has less than 3 dimensions.

5.87.3.11 template<typename T> const T& repast::Point< T >::operator[] (size_t *index*) const `[inline]`

Gets the coordinate value at the specified index.

Parameters

index	the dimension of the point to get the coordinate for. X is the first, y is the second and so on.

Returns

the coordinate of the point in the specified dimension.

5.87.3.12 template<typename T> T& repast::Point< T >::operator[] (size_t *index*) [inline]

Gets the coordinate value at the specified index.

Parameters

index	the dimension of the point to get the coordinate for. X is the first, y is the second and so on.

Returns

the coordinate of the point in the specified dimension.

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

- repast_hpc/Point.h

5.88 repast::ProbItem Class Reference

Helper class for calculating outcomes based on a set of probabilities that sum to 1.

`#include <NetworkBuilder.h>`

Public Member Functions

- **ProbItem** (int i, double lb, double ub)
- bool **contains** (double val)
- int **index** () const

5.88.1 Detailed Description

Helper class for calculating outcomes based on a set of probabilities that sum to 1.

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

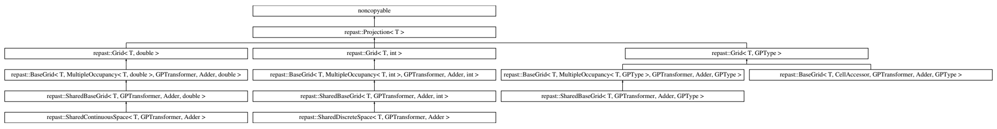
- repast_hpc/NetworkBuilder.h
- repast_hpc/NetworkBuilder.cpp

5.89 repast::Projection< T > Class Template Reference

Abstract base class for all Projections.

`#include <Projection.h>`

Inheritance diagram for repast::Projection< T >:


noncopyable
repast::Projection< T >
repast::Grid< T, double >
repast::Grid< T, int >
repast::Grid< T, GPType >
repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >
repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >
repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >
repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >
repast::SharedBaseGrid< T, GPTransformer, Adder, double >
repast::SharedBaseGrid< T, GPTransformer, Adder, int >
repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >
repast::SharedContinuousSpace< T, GPTransformer, Adder >
repast::SharedDiscreteSpace< T, GPTransformer, Adder >

Public Types

- enum **RADIUS** { **PRIMARY**, **SECONDARY** }

Public Member Functions

- Projection (std::string name)

  Creates a projection with specified name.
- const std::string name () const

  Gets the name of this projection.
- void addFilterVal (int type)

  Adds an entry to the list of agent types that can be added to this projection.
- void removeFilterVal (int type)

  Removes an entry from the list of agent types that can be added to this projection.
- void clearFilter ()

  Clears the list of agent types that can be added to this projection; the result is that the filter is empty, and any agent can be added.
- bool agentCanBeAdded (boost::shared_ptr< T > agent)

  Returns true if the agent can be added to the projection, which will be the case if the filter list is empty or if the agent's type is in the filter list.
- virtual bool keepsAgentsOnSyncProj ()=0

  Should return true if the Projection implemented can 'keep' some (non-local) agents during a projection information synchronization operation.
- virtual bool sendsSecondaryAgentsOnStatusExchange ()=0

  Should return true if the Projection implemented will send secondary agents during a status exchange.
- virtual void getInfoExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceiveFrom)=0

  Gets the set of processes with which this Projection exchanges projection info.
- virtual void getAgentStatusExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceive-From)=0

  Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.
- virtual void getRequiredAgents (std::set< AgentId > &agentsToTest, std::set< AgentId > &agentsRequired, RADIUS radius=PRIMARY)=0

  Given a set of agents to test, gets the subset that must be kept in order to fulfill the projection's 'contract' to the specified radius.
- virtual void getAgentsToPush (std::set< AgentId > &agentsToTest, std::map< int, std::set< AgentId > > &agentsToPush)=0

  Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.
- virtual void getProjectionInfo (std::vector< AgentId > &agents, std::vector< ProjectionInfoPacket ∗ > &pack-ets, bool secondaryInfo=false, std::set< AgentId > ∗secondaryIds=0, int destProc=-1)

  Convenience wrapper that gets all of the projection information for the agents specified (calls implementation in child class that gets only the information for one agent).
- void updateProjectionInfo (std::vector< ProjectionInfoPacket ∗ > &pips, Context< T > ∗context)

  Updates the projection information for the agents in this projection according to the information contained in the vector of information packets passed.
- virtual void **cleanProjectionInfo** (std::set< AgentId > &agentsToKeep)=0
- virtual void **balance** ()

Protected Member Functions

- virtual bool **addAgent** (boost::shared_ptr< T > agent)=0
- virtual void **removeAgent** (T ∗agent)=0
- virtual ProjectionInfoPacket ∗ **getProjectionInfo** (AgentId id, bool secondaryInfo=false, std::set< AgentId > ∗secondaryIds=0, int destProc=-1)=0
- virtual void **updateProjectionInfo** (ProjectionInfoPacket ∗pip, Context< T > ∗context)=0

Protected Attributes

- std::string **name_**
- std::set< int > **filter**

Friends

- class **Context**< **T** >

5.89.1 Detailed Description

template<typename T>class repast::Projection< T >

Abstract base class for all Projections.

5.89.2 Constructor & Destructor Documentation

5.89.2.1 template<typename T> repast::Projection< T >::Projection (std::string *name*) [inline]

Creates a projection with specified name.

Parameters

name	the name of the projection. This must be unique across projections

5.89.3 Member Function Documentation

5.89.3.1 template<typename T> void repast::Projection< T >::addFilterVal (int *type*) [inline]

Adds an entry to the list of agent types that can be added to this projection.

Note: no indication if type is already listed

Parameters

type	type to be added

5.89.3.2 template<typename T> bool repast::Projection< T >::agentCanBeAdded (boost::shared_ptr< T > *agent*) [inline]

Returns true if the agent can be added to the projection, which will be the case if the filter list is empty or if the agent's type is in the filter list.

Parameters

agent	pointer to the agent to be tested

5.89.3.3 template<typename T> virtual void **repast::Projection**< T >::getAgentStatusExchangePartners (std::set< int > & *psToSendTo*, std::set< int > & *psToReceiveFrom*) `[pure virtual]`

Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.

In the most general case this will be all other processors. However, simulations where agents move in spaces will usually exchange agents only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, repast::SharedBaseGrid< T, GPTransformer, Adder, double >, repast::Graph< V, E, Ec, EcM >, repast::Grid< T, GPType >, repast::Grid< T, double >, and repast::Grid< T, int >.

5.89.3.4 template<typename T> virtual void **repast::Projection**< T >::getAgentsToPush (std::set< **AgentId** > & *agentsToTest*, std::map< int, std::set< **AgentId** > > & *agentsToPush*) `[pure virtual]`

Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.

Generally spaces must push agents that are in 'buffer zones' and graphs must push local agents that are vertices to master edges where the other vertex is non- local. The results are returned per-process in the agentsToPush map.

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, repast::SharedBaseGrid< T, GPTransformer, Adder, double >, repast::BaseGrid< T, CellAccessor, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >, repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >, repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >, repast::Graph< V, E, Ec, EcM >, repast::Grid< T, GPType >, repast::Grid< T, double >, repast::Grid< T, int >, and repast::SharedDiscreteSpace< T, GPTransformer, Adder >.

5.89.3.5 template<typename T> virtual void **repast::Projection**< T >::getInfoExchangePartners (std::set< int > & *psToSendTo*, std::set< int > & *psToReceiveFrom*) `[pure virtual]`

Gets the set of processes with which this Projection exchanges projection info.

In the most general case this will be all other processors; this is the case for graphs, where agent connections can be arbitrary. However, spaces usually exchange information only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implemented in repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >, repast::SharedBaseGrid< T, GPTransformer, Adder, int >, repast::SharedBaseGrid< T, GPTransformer, Adder, double >, repast::Graph< V, E, Ec, EcM >, repast::Grid< T, GPType >, repast::Grid< T, double >, and repast::Grid< T, int >.

5.89.3.6 template<typename T> virtual void **repast::Projection**< T >::getRequiredAgents (std::set< **AgentId** > & *agentsToTest*, std::set< **AgentId** > & *agentsRequired*, RADIUS *radius* = `PRIMARY`) `[pure virtual]`

Given a set of agents to test, gets the subset that must be kept in order to fulfill the projection's 'contract' to the specified radius.

Generally spaces do not require any agents, but graphs do- generally the non-local ends to master copies of edges.

Implemented in repast::Grid< T, GPType >, repast::Grid< T, double >, and repast::Grid< T, int >.

5.89.3.7 template<typename T> virtual bool repast::Projection< T >::keepsAgentsOnSyncProj ( ) [pure virtual]

Should return true if the Projection implemented can 'keep' some (non-local) agents during a projection information synchronization operation.

Generally spaces will allow all non-local agents to be deleted, but graphs keep the non-local agents that participate in Master edges.

It is possible to override these. A graph projection can be created that does not permit non-local agents to be 'kept'. This would be an extremely unusual use case, but it is possible.

Note that these are used for optimization. If no projection in a given context keeps any agents, several steps in the synchronization algorithm can be omitted. Of course, omitting these steps when a projection actually retains agents can caused undefined problems.

Returns

true if this projection will keep non-local agents during a projection information synchronziation event, false if it will not.

Implemented in repast::Graph< V, E, Ec, EcM >, repast::Grid< T, GPType >, repast::Grid< T, double >, and repast::Grid< T, int >.

5.89.3.8 template<typename T> void repast::Projection< T >::removeFilterVal (int *type*) [inline]

Removes an entry from the list of agent types that can be added to this projection.

Note: no indication if type is not listed

Parameters

type	entry to be removed

5.89.3.9 template<typename T> virtual bool repast::Projection< T >::sendsSecondaryAgentsOnStatusExchange ( ) [pure virtual]

Should return true if the Projection implemented will send secondary agents during a status exchange.

Generally spaces do not and graphs do.

If no secondary agents will be sent, portions of the algorithm can be omitted for optimization.

Returns

true if the Projection returns secondary agents, false if not

Implemented in repast::Graph< V, E, Ec, EcM >, repast::Grid< T, GPType >, repast::Grid< T, double >, and repast::Grid< T, int >.

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

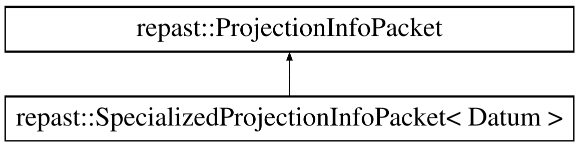
- repast_hpc/Projection.h

5.90 repast::ProjectionInfoPacket Class Reference

Serializable packet that can contain projection information regardless of the type of projection (network or spatial).

`#include <Projection.h>`

Inheritance diagram for repast::ProjectionInfoPacket:


repast::ProjectionInfoPacket
repast::SpecializedProjectionInfoPacket< Datum >

Public Member Functions

- **ProjectionInfoPacket** (AgentId agentId)
- template<class Archive >
  void **serialize** (Archive &ar, const unsigned int version)
- virtual bool **isEmpty** ()

Public Attributes

- AgentId **id**

Friends

- class **boost::serialization::access**

5.90.1 Detailed Description

Serializable packet that can contain projection information regardless of the type of projection (network or spatial).

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

- repast_hpc/Projection.h

5.91 repast::Properties Class Reference

Map type object that contains key, value(string) properties.

```
#include <Properties.h>
```

Public Types

- typedef
  boost::transform_iterator
  < KeyGetter, std::map
  < std::string, std::string >
  ::const_iterator > **key_iterator**

Public Member Functions

- Properties ()

  Creates an empty Properties.
- Properties (const std::string &file, boost::mpi::communicator ∗comm=0, int maxPropFileSize=MAX_PROP_-FILE_SIZE)

  Creates a new Properties using the properties defined in the specified file.

- Properties (const std::string &file, int argc, char $**$argv, boost::mpi::communicator $*$comm=0, int maxPropFileSize=MAX_PROP_FILE_SIZE)

  Creates a new Properties using the properties defined in the specified file and any properties specified in Key=Val format in the argument array.
- Properties (int argc, char $**$argv)

  Creates a new Properties using the properties specified in Key=Val format in the argument.
- void putProperty (const std::string &key, std::string value)

  Puts a property into this Properties with the specified key and value.
- void putProperty (const std::string &key, long double value)

  Puts a property into this Properties with the specified key and value.
- std::string getProperty (const std::string &key) const

  Gets the property with the specified key.
- bool contains (const std::string &key) const

  Gets whether or not this Properties contains the specified key.
- key_iterator keys_begin () const

  Gets the start of an iterator over this Properties' keys.
- key_iterator keys_end () const

  Gets the end of an iterator over this Properties' keys.
- void readFile (const std::string &file, boost::mpi::communicator $*$comm=0, int maxPropFileSize=MAX_PROP_FILE_SIZE)

  Adds any properties defined in the specified file.
- void processCommandLineArguments (int argc, char $**$argv)

  Processes a char$**$ array of the given size; any component that has an equals sign is entered as a property value, overriding any previous entry read from the properies file.
- int size () const

  Gets the number of properties in this Properties.
- void log (std::string logName, std::vector$<$ std::string $>$ $*$keysToWrite=0)

  Writes the contents of the properties file to the specified repast log (at 'INFO' log level)
- void writeToSVFile (std::string fileName, std::string separator=",")

  Writes the contents of the properties file to the specified separated-value file.
- void writeToSVFile (std::string fileName, std::vector$<$ std::string $>$ &keysToWrite, std::string separator=",")

  Writes the contents of the properties file to the specified separated-value file.

5.91.1 Detailed Description

Map type object that contains key, value(string) properties.

A Properties instance can be constructed from a file. Each line is a property with the key and value separated by =. For example,

some.property = 3

another.property = hello

5.91.2 Constructor & Destructor Documentation

5.91.2.1 repast::Properties::Properties (const std::string & *file*, boost::mpi::communicator $*$ *comm* = `0`, int *maxPropFileSize* = `MAX_PROP_FILE_SIZE`)

Creates a new Properties using the properties defined in the specified file.

Each line is a property with the key and value separated by =. For example,

some.property = 3

another.property = hello

Parameters

file	the properties file path
comm	pointer to a communicator; if null (the default), all processes read the properties file separately. If a communicator is provided, rank 0 reads the file and broadcasts it to all other ranks.
maxPropFileSize	optional parameter; if the properties file is larger than the default MAX_PROP_FILE_SIZE, the new size can be passed here.

5.91.2.2 repast::Properties::Properties (const std::string & *file*, int *argc*, char ∗∗ *argv*, boost::mpi::communicator ∗ *comm* = `0`, int *maxPropFileSize* = `MAX_PROP_FILE_SIZE`)

Creates a new Properties using the properties defined in the specified file and any properties specified in Key=Val format in the argument array.

Properties in the argument array will supersede any in the properties file.

Each line in the properties file is a property with the key and value separated by =. For example,

some.property = 3

another.property = hello

Parameters

file	the properties file path
argc	count of the elements in the argv array
array	of char∗ that may include Key=Value pairs. Elements with no '=' are ignored.
comm	pointer to a communicator; if null (the default), all processes read the properties file separately. If a communicator is provided, rank 0 reads the file and broadcasts it to all other ranks.
maxPropFileSize	optional parameter; if the properties file is larger than the default MAX_PROP_FILE_SIZE, the new size can be passed here.

5.91.2.3 repast::Properties::Properties (int *argc*, char ∗∗ *argv*)

Creates a new Properties using the properties specified in Key=Val format in the argument.

Parameters

argc	count of the elements in the argv array
array	of char∗ that may include Key=Value pairs. Elements with no '=' are ignored.

5.91.3 Member Function Documentation

5.91.3.1 bool repast::Properties::contains (const std::string & *key*) const

Gets whether or not this Properties contains the specified key.

Parameters

key	the property key

5.91.3.2 string repast::Properties::getProperty (const std::string & *key*) const

Gets the property with the specified key.

Parameters

key	the property key

Returns

the value for that key, or an empty string if the property is not found.

5.91.3.3 key_iterator repast::Properties::keys_begin ( ) const [inline]

Gets the start of an iterator over this Properties' keys.

Returns

the start of an iterator over this Properties' keys.

5.91.3.4 key_iterator repast::Properties::keys_end ( ) const [inline]

Gets the end of an iterator over this Properties' keys.

Returns

the end of an iterator over this Properties' keys.

5.91.3.5 void repast::Properties::log (std::string *logName*, std::vector$<$ std::string $>$ $*$ *keysToWrite* = 0) [inline]

Writes the contents of the properties file to the specified repast log (at 'INFO' log level)

Parameters

logName	name of the log to use
keysToWrite	optional; if included, writes only the keys included in the vector and their values, in the order they appear in the vector. Will write blank values for any key name in the vector that is not in the properties file. If not included, all properties and their values are written, in map order.

5.91.3.6 void repast::Properties::processCommandLineArguments (int *argc*, char $**$ *argv*)

Processes a char$**$ array of the given size; any component that has an equals sign is entered as a property value, overriding any previous entry read from the properies file.

Parameters

argc	the number of entries in the array
argv	the array of char values to be mapped

5.91.3.7 void repast::Properties::putProperty (const std::string & *key*, std::string *value*)

Puts a property into this Properties with the specified key and value.

Parameters

key	the property key
value	the property value

5.91.3.8 void repast::Properties::putProperty (const std::string & *key,* long double *value*)

Puts a property into this Properties with the specified key and value.

Note that even though the second argument can be passed as a numeric value, it is stored as a string

Parameters

key	the property key
value	the property value

5.91.3.9 void repast::Properties::readFile (const std::string & *file,* boost::mpi::communicator ∗ *comm* = 0, int *maxPropFileSize* = MAX_PROP_FILE_SIZE)

Adds any properties defined in the specified file.

Each line is a property with the key and value separated by =. For example,

some.property = 3

another.property = hello

Parameters

file	the properties file path
comm	pointer to a communicator; if null (the default), all processes read the properties file separately. If a communicator is provided, rank 0 reads the file and broadcasts it to all other ranks.

5.91.3.10 int repast::Properties::size ( ) const [inline]

Gets the number of properties in this Properties.

Returns

the number of properties in this Properties.

5.91.3.11 void repast::Properties::writeToSVFile (std::string *fileName,* std::string *separator* = " , ")

Writes the contents of the properties file to the specified separated-value file.

If the file does not exist it is created and a header line is written with the key values.

Parameters

fileName	name

5.91.3.12 void repast::Properties::writeToSVFile (std::string *fileName,* std::vector< std::string > & *keysToWrite,* std::string *separator* = " , ")

Writes the contents of the properties file to the specified separated-value file.

If the file does not exist it is created and a header line is written with the key values.

Parameters

fileName	name

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

- repast_hpc/Properties.h
- repast_hpc/Properties.cpp

5.92 repast::Random Class Reference

Methods for working with random distributions, draws etc.

`#include <Random.h>`

Public Member Functions

- void putGenerator (const std::string &id, NumberGenerator ∗generator)

  Puts the named generator into this Random.
- NumberGenerator ∗ getGenerator (const std::string &id)

  Gets the named generator or 0 if the name is not found.
- boost::mt19937 & engine ()

  Gets the random number engine from which the distributions are created.
- boost::uint32_t seed ()

  Gets the current seed.
- double nextDouble ()

  Gets the next double in the range [0, 1).
- DoubleUniformGenerator createUniDoubleGenerator (double from, double to)

  Creates a generator that produces doubles in the range [from, to).
- IntUniformGenerator createUniIntGenerator (int from, int to)

  Creates a generator that produces ints in the range [from, to].
- TriangleGenerator createTriangleGenerator (double lowerBound, double mostLikely, double upperBound)

  Creates a triangle generator with the specified properties.
- CauchyGenerator createCauchyGenerator (double median, double sigma)

  pdf: p(x) = sigma/(pi∗(sigma∗∗2 + (x-median)∗∗2))
- ExponentialGenerator createExponentialGenerator (double lambda)

  pdf: p(x) = lambda ∗ exp(-lambda ∗ x)
- NormalGenerator createNormalGenerator (double mean, double sigma)

  Creates a normal generator.
- LogNormalGenerator createLogNormalGenerator (double mean, double sigma)

  Produces random numbers with p(x) = 1/(x ∗ normal_sigma ∗ sqrt(2∗pi)) ∗ exp(-(log(x)-normal_mean)2 / (2∗normal-_sigma2)) for $x > 0$, where normal_mean = log(mean2/sqrt(sigma2 + mean2)) and normal_sigma = sqrt(log(1 + sigma2/mean2))

Static Public Member Functions

- static void initialize (boost::uint32_t seed)

  Initialize the Random singleton with the specified seed.
- static Random ∗ instance ()

  Gets the singleton instance of this Random.

Protected Member Functions

- **Random** (boost::uint32_t seed)

5.92.1 Detailed Description

Methods for working with random distributions, draws etc.

5.92.2 Member Function Documentation

5.92.2.1 CauchyGenerator repast::Random::createCauchyGenerator (double *median,* double *sigma*)

pdf: p(x) = sigma/(pi∗(sigma∗∗2 + (x-median)∗∗2))

Parameters

median	
sigma	

Returns

a Cauchy generator.

5.92.2.2 ExponentialGenerator repast::Random::createExponentialGenerator (double *lambda*)

pdf: p(x) = lambda ∗ exp(-lambda ∗ x)

Parameters

lambda	must be $>$ 0

Returns

an exponential generator.

5.92.2.3 NormalGenerator repast::Random::createNormalGenerator (double *mean,* double *sigma*)

Creates a normal generator.

pdf: p(x) = 1/sqrt(2∗pi∗sigma) ∗ exp(- (x-mean)2 / (2∗sigma2))

5.92.2.4 TriangleGenerator repast::Random::createTriangleGenerator (double *lowerBound,* double *mostLikely,* double *upperBound*)

Creates a triangle generator with the specified properties.

A TriangleGenerator produces a floating point value x where lowerbound $<=$ x $<=$ upperBound and mostLikely is the most probable value for x.

Parameters

lowerBound	the lower bound of the values produced by the generator
mostLikely	the most likely value produced by the generator
upperBound	the upper bound of the values produced by the generator

Returns

a triangle generator.

5.92.2.5 DoubleUniformGenerator repast::Random::createUniDoubleGenerator (double *from,* double *to*)

Creates a generator that produces doubles in the range [from, to).

inclusive of from, exclusive of to.

Parameters

from	the range start (inclusive)
to	the range end (exclusive)

Returns

a generator that produces doubles in the range [from, to).

5.92.2.6 IntUniformGenerator repast::Random::createUniIntGenerator (int *from,* int *to*)

Creates a generator that produces ints in the range [from, to].

Parameters

from	the range start (inclusive)
to	the range end (inclusive)

Returns

a generator that produces ints in the range [from, to].

5.92.2.7 boost::mt19937& repast::Random::engine ( ) [inline]

Gets the random number engine from which the distributions are created.

Returns

he random number engine from which the distributions are created.

5.92.2.8 NumberGenerator ∗ repast::Random::getGenerator (const std::string & *id*)

Gets the named generator or 0 if the name is not found.

Parameters

id	the name of the generator to get

5.92.2.9 void repast::Random::initialize (boost::uint32_t *seed*) [static]

Initialize the Random singleton with the specified seed.

Parameters

the	seed to initialize the random number generator with.

5.92.2.10 double repast::Random::nextDouble ( )

Gets the next double in the range [0, 1).

Returns

the next double in the range [0, 1).

5.92.2.11 void repast::Random::putGenerator (const std::string & *id,* NumberGenerator ∗ *generator*)

Puts the named generator into this Random.

Added generators will be deleted by Random when it is destroyed.

Parameters

the	id of the generator
generator	the generator to add

5.92.2.12 boost::uint32_t repast::Random::seed ( ) [inline]

Gets the current seed.

Returns

the current seed.

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

- repast_hpc/Random.h
- repast_hpc/Random.cpp

5.93 repast::RandomAccess< I > Class Template Reference

Given an iterator and a number of elements, creates a data structure that allows efficient access to those elements.

`#include <Random.h>`

Public Member Functions

- RandomAccess (I beginning, int size)

  Constructs a RandomAccess instance for this iterator.
- I get (int index)

  Gets the element at the specified index.

5.93.1 Detailed Description

template<typename I>class repast::RandomAccess< I >

Given an iterator and a number of elements, creates a data structure that allows efficient access to those elements.

Is only valid as long as the iterator is valid.

The basic implementation creates a vector of ordered pairs linking an integer and an iterator pointing to an element in the original iteration set. To find the nth element, the algorithm searches backwards through the list of 'landmarks', finds the highest landmark lower than n, chooses the iterator associated with that landmark, and steps forward until n is reached, adding new landmarks if appropriate. So given landmarks:

0 - pointer to element 0 100 - pointer to element 100 200 - pointer to element 200

if the request for element 438 is given, the algorithm will search backward and find landmark 200; it will then step forward, adding landmarks for 300 and 400, until element 438 is reached and returned.

Assuming that requests are evenly distributed, optimum interval for landmarks is the square root of the size of the list, and performance for the algorithm will be in log(size) time.

Note that other implementations are possible- for example, checking if enough memory would allow a completely indexed list. A long-term possibility is allowing the user to specify (for example, specify that the algorithm with lowest memory cost be used even though memory is initially available, perhaps because other routines will be filling that memory while this object is in use).

5.93.2 Constructor & Destructor Documentation

5.93.2.1 template<typename I > repast::RandomAccess< I >::RandomAccess (I *beginning*, int *size*) `[inline]`

Constructs a RandomAccess instance for this iterator.

Parameters

beginning	
size	

5.93.3 Member Function Documentation

5.93.3.1 template<typename I > I repast::RandomAccess< I >::get (int *index*) `[inline]`

Gets the element at the specified index.

Parameters

index	

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

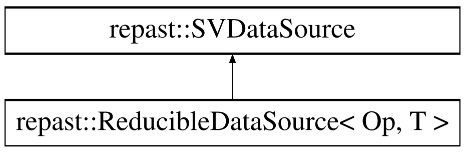
- repast_hpc/Random.h

5.94 repast::ReducibleDataSource< Op, T > Class Template Reference

Source of data and a reduction operation.

`#include <ReducibleDataSource.h>`

Inheritance diagram for repast::ReducibleDataSource< Op, T >:



Public Member Functions

- **ReducibleDataSource** (std::string name, TDataSource< T > *dataSource, Op op)
- virtual void **record** ()
- virtual void **write** (Variable *var)
- virtual SVDataSource::DataType **type** () const

Protected Attributes

- Op **_op**

- std::vector< T > **data**
- TDataSource< T > ∗ **_dataSource**
- int **rank**

5.94.1 Detailed Description

template<typename Op, typename T>class repast::ReducibleDataSource< Op, T >

Source of data and a reduction operation.

Used internally by a SVDataSet to store the data sources. their associated ops etc.

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

- repast_hpc/ReducibleDataSource.h

5.95 repast::RepastEdge< V > Class Template Reference

Default graph / network edge implementation.

```
#include <Edge.h>
```

Public Types

- enum **MASTER_NODE** { **DEFAULT**, **SOURCE**, **TARGET** }

Public Member Functions

- RepastEdge (V ∗source, V ∗target, MASTER_NODE useTargetAsMaster=DEFAULT)

  Creates a RepastEdge with the specified source and target and a default weight of 1.
- RepastEdge (V ∗source, V ∗target, double weight, MASTER_NODE useTargetAsMaster=DEFAULT)

  Creates a RepastEdge with the specified source, target, and weight.
- RepastEdge (boost::shared_ptr< V > source, boost::shared_ptr< V > target, MASTER_NODE useTarget-AsMaster=DEFAULT)

  Creates a RepastEdge with the specified source and target and a default weight of 1.
- RepastEdge (boost::shared_ptr< V > source, boost::shared_ptr< V > target, double weight, MASTER_N-ODE useTargetAsMaster=DEFAULT)

  Creates a RepastEdge with the specified source, target, and weight.
- RepastEdge (const RepastEdge &edge)

  Copy constructor that creates a RepastEdge from another RepastEdge.
- V ∗ source () const

  Gets the source of this RepastEdge.
- V ∗ target () const

  Gets the target of this RepastEdge.
- void **target** (V ∗target)
- void **source** (V ∗source)
- double weight () const

  Gets the weight of this RepastEdge.
- void **weight** (double wt)
- bool **usesTargetAsMaster** ()
- void **markConflicted** ()
- void **clearConflicted** ()
- bool **isConflicted** ()

5.95.1 Detailed Description

template<typename V>class repast::RepastEdge< V >

Default graph / network edge implementation.

Template Parameters

V	agent type that is the source and target of the edge

5.95.2 Constructor & Destructor Documentation

5.95.2.1 template<typename V > repast::RepastEdge< V >::RepastEdge (V * *source,* V * *target,* MASTER_NODE *useTargetAsMaster* = `DEFAULT`)

Creates a RepastEdge with the specified source and target and a default weight of 1.

Parameters

source	the edge source
target	the edge target

5.95.2.2 template<typename V > repast::RepastEdge< V >::RepastEdge (V * *source,* V * *target,* double *weight,* MASTER_NODE *useTargetAsMaster* = `DEFAULT`)

Creates a RepastEdge with the specified source, target, and weight.

Parameters

source	the edge source
target	the edge target
weight	the edge weight

5.95.2.3 template<typename V > repast::RepastEdge< V >::RepastEdge (boost::shared_ptr< V > *source,* boost::shared_ptr< V > *target,* MASTER_NODE *useTargetAsMaster* = `DEFAULT`)

Creates a RepastEdge with the specified source and target and a default weight of 1.

Parameters

source	the edge source
target	the edge target

5.95.2.4 template<typename V > repast::RepastEdge< V >::RepastEdge (boost::shared_ptr< V > *source,* boost::shared_ptr< V > *target,* double *weight,* MASTER_NODE *useTargetAsMaster* = `DEFAULT`)

Creates a RepastEdge with the specified source, target, and weight.

Parameters

source	the edge source
target	the edge target
weight	the edge weight

5.95.3 Member Function Documentation

5.95.3.1 template<typename V> V* repast::RepastEdge< V >::source ( ) const [inline]

Gets the source of this RepastEdge.

Returns

the source of this RepastEdge.

5.95.3.2 template<typename V> V* repast::RepastEdge< V >::target ( ) const [inline]

Gets the target of this RepastEdge.

Returns

the target of this RepastEdge.

5.95.3.3 template<typename V> double repast::RepastEdge< V >::weight ( ) const [inline]

Gets the weight of this RepastEdge.

Returns

the weight of this RepastEdge.

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

- repast_hpc/Edge.h

5.96 repast::RepastEdgeContent< V > Struct Template Reference

Serializable; also, does not include agent content, only agent IDs.

`#include <Edge.h>`

Public Member Functions

- template<class Archive >
  void **serialize** (Archive &ar, const unsigned int version)
- **RepastEdgeContent** (RepastEdge< V > *edge)

Public Attributes

- AgentId **source**
- AgentId **target**
- double **weight**
- bool **usesTargetAsMaster**

Friends

- class **boost::serialization::access**

5.96.1 Detailed Description

template<typename V>struct repast::RepastEdgeContent< V >

Serializable; also, does not include agent content, only agent IDs.

Template Parameters

V	type for vertices; must provide AgentID

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

- repast_hpc/Edge.h

5.97 repast::RepastEdgeContentManager< V > Class Template Reference

Class for creating RepastEdges from RepastEdgeContent, and vice versa.

`#include <Edge.h>`

Public Member Functions

- RepastEdge< V > ∗ **createEdge** (RepastEdgeContent< V > &content, Context< V > ∗context)
- RepastEdgeContent< V > ∗ **provideEdgeContent** (RepastEdge< V > ∗edge)

5.97.1 Detailed Description

template<typename V>class repast::RepastEdgeContentManager< V >

Class for creating RepastEdges from RepastEdgeContent, and vice versa.

Template Parameters

V	type for vertices; must provide AgentID

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

- repast_hpc/Edge.h

5.98 repast::RepastEvent Class Reference

General class linking a function pointer to a specific tick.

`#include <Schedule.h>`

Public Attributes

- double **tick**
- boost::shared_ptr< Functor > **func_ptr**

5.98.1 Detailed Description

General class linking a function pointer to a specific tick.

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

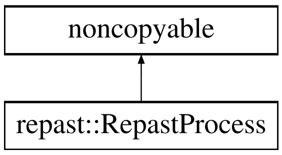
- repast_hpc/Schedule.h
- repast_hpc/Schedule.cpp

5.99 repast::RepastProcess Class Reference

Encapsulates the process in which repast is running and manages interprocess communication etc.

`#include <RepastProcess.h>`

Inheritance diagram for repast::RepastProcess:



Public Types

- enum **EXCHANGE_PATTERN** { **POLL**, **USE_CURRENT**, **USE_LAST_OR_POLL**, **USE_LAST_OR_USE_-CURRENT** }

Public Member Functions

- void agentRemoved (const AgentId &id)

  NON USER API.
- void moveAgent (const AgentId &id, int process)

  NON USER API.
- void addExportedAgent (int importingProcess, AgentId id)

  NON USER API.
- void addImportedAgent (AgentId id)

  NON USER API.
- int rank () const

  Gets the rank of this process.
- int worldSize () const

  Gets the number of processes in the world.
- void done ()

  Notifes this RepastProcess that simulation has completed.
- ScheduleRunner & getScheduleRunner ()

  Gets the ScheduleRunner used by this RepastProcess.
- boost::mpi::communicator ∗ **getCommunicator** ()
- void **dropImporterExporterSet** (std::string setName)
- std::string **ImporterExporterVersion** ()
- std::string **ImporterExporterReport** ()
- template<typename T , typename Content , typename Provider , typename Updater , typename AgentCreator >
  void requestAgents (SharedContext< T > &context, AgentRequest &request, Provider &provider, Updater &updater, AgentCreator &creator, std::string setName=DEFAULT_AGENT_REQUEST_SET, AGENT_IMP-ORTER_EXPORTER_TYPE setType=DEFAULT_ENUM_SYMBOL)

  Request agents from other processes.
- template<typename Content , typename Provider , typename Updater >
  void synchronizeAgentStates (Provider &provider, Updater &updater, std::string setName=REQUEST_AGE-NTS_ALL)

  Synchronizes the state values of shared agents.
- template<typename T , typename Content , typename Provider , typename Updater , typename AgentCreator >
  void synchronizeProjectionInfo (SharedContext< T > &context, Provider &provider, Updater &updater, AgentCreator &creator, EXCHANGE_PATTERN exchangePattern=POLL, bool declareNoAgentsKeptOnAny-Process=false)

Synchronizes the Projection information for shared projections.

- template<typename T , typename Content , typename Provider , typename AgentCreator , typename Updater >
  void synchronizeAgentStatus (SharedContext< T > &context, Provider &provider, Updater &updater, AgentCreator &creator, EXCHANGE_PATTERN exchangePattern=POLL)

  Synchronizes the status (moved or died) of all agents across processes.

Static Public Member Functions

- static RepastProcess ∗ init (std::string propsfile, boost::mpi::communicator ∗comm=0, int maxConfigFileSize=MAX_CONFIG_FILE_SIZE)

  Initialize this RepastProcess.
- static RepastProcess ∗ instance ()

  Gets this RepastProcess.
- static boost::mpi::communicator ∗ **communicator** ()

Protected Member Functions

- **RepastProcess** (boost::mpi::communicator ∗comm=0)
- void **saveProjInfoSRProcs** (std::vector< int > &sends, std::vector< int > &recvs)
- void **saveAgentStatusInfoSRProcs** (std::vector< int > &sends, std::vector< int > &recvs)

5.99.1 Detailed Description

Encapsulates the process in which repast is running and manages interprocess communication etc.

This is singleton to insure that there is one per actual process.

5.99.2 Member Function Documentation

5.99.2.1 void repast::RepastProcess::addExportedAgent (int *importingProcess,* AgentId *id*)

NON USER API.

Notifies this RepastProcess that it is exporting the specified agent to the specified process. This sort of notification is done automatically when requesting agents, but agents may get added in other ways.

5.99.2.2 void repast::RepastProcess::addImportedAgent (AgentId *id*)

NON USER API.

Notifies this RepastProcess that it is importing the specified agent. This sort of notification is normally done automatically when requesting agents, but imports can occur in other ways.

5.99.2.3 void repast::RepastProcess::agentRemoved (const AgentId & *id*)

NON USER API.

Notifies this RepastProcess that the specified agent has been removed (e.g. the agent "died").

5.99.2.4 void repast::RepastProcess::done ( )

Notifes this RepastProcess that simulation has completed.

This should be called when the simulation has completed.

5.99.2.5 ScheduleRunner& repast::RepastProcess::getScheduleRunner ( ) [inline]

Gets the ScheduleRunner used by this RepastProcess.

Returns

the ScheduleRunner used by this RepastProcess.

5.99.2.6 RepastProcess ∗ repast::RepastProcess::init (std::string *propsfile*, boost::mpi::communicator ∗ *comm* = 0, int *maxConfigFileSize* = MAX_CONFIG_FILE_SIZE) [static]

Initialize this RepastProcess.

This must be called before the RepastProcess is used. If a configuration properties file is specified this properties file will be used to configure logging.

Parameters

propsfile	a configuration properties file. This can be an empty string.

5.99.2.7 RepastProcess ∗ repast::RepastProcess::instance ( ) [static]

Gets this RepastProcess.

Returns

this RepastProcess instance.

5.99.2.8 void repast::RepastProcess::moveAgent (const AgentId & *id*, int *process*)

NON USER API.

Notifies this RepastProcess that the specified agent should be moved from this process to the specified process.

Parameters

id	the id of the agent to be moved
process	the process to move the agent to

5.99.2.9 int repast::RepastProcess::rank ( ) const [inline]

Gets the rank of this process.

Returns

the rank of this process.

5.99.2.10 template<typename T , typename Content , typename Provider , typename Updater , typename AgentCreator > void repast::RepastProcess::requestAgents (SharedContext< T > & *context*, AgentRequest & *request*, Provider & *provider*, Updater & *updater*, AgentCreator & *creator*, std::string *setName* = DEFAULT_AGENT_REQUEST_SET, AGENT_IMPORTER_EXPORTER_TYPE *setType* = DEFAULT_ENUM_SYMBOL)

Request agents from other processes.

Requests agents from one process to others.

Copies of the requested agents' Content are retrieved from their respective processes, created using the Agent-Creator and added to the specified context.

Parameters

context	the context to which the requested agents will be added
request	the AgentRequest containing the ids of the requested agents
provider	provides Content for a given an AgentRequest
creator	creates agents of type T given Content.

Template Parameters

T	the type of the agents in the context
Content	the serializable struct or class that describes the state of agents
Provider	given an AgentRequest, a Provider provides the Content for the requested agents, implementing void provideContent(const AgentRequest&, std::vector<-Content>&)
AgentCreator	a class that can create agents from Content, implementing T∗ createAgent(-Content&).

5.99.2.11 template<typename Content , typename Provider , typename Updater > void repast::RepastProcess::synchronize-AgentStates (Provider & *provider,* Updater & *updater,* std::string *setName* = `REQUEST_AGENTS_ALL`)

Synchronizes the state values of shared agents.

Does not change the Projection information for those agents.

5.99.2.12 template<typename T , typename Content , typename Provider , typename AgentCreator , typename Updater > void repast::RepastProcess::synchronizeAgentStatus (SharedContext< T > & *context,* Provider & *provider,* Updater & *updater,* AgentCreator & *creator,* EXCHANGE_PATTERN *exchangePattern* = `POLL`)

Synchronizes the status (moved or died) of all agents across processes.

Parameters

context	the SharedContext that contains the agents on this proceses
provider	the class that provides agents given an AgentRequest
creator	creates agents of type T given Content.

Template Parameters

T	the type of agents in the context
Content	the serializable struct or class that describes an agents state.
Provider	a class that provides Content, when given an AgentRequest, implementing void provideContent(const repast::AgentRequest&, std::vector<Content>& out)
AgentCreator	a class that can create agents from Content, implementing T∗ createAgent(-Content&).

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

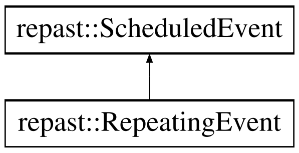
- repast_hpc/RepastProcess.h
- repast_hpc/RepastProcess.cpp

5.100 repast::RepeatingEvent Class Reference

ScheduledEvent that executes repeatedly.

`#include <Schedule.h>`

Inheritance diagram for repast::RepeatingEvent:



Public Member Functions

- **RepeatingEvent** (double start, double _interval, RepastEvent $*$)
- virtual bool reschedule (std::priority_queue$<$ ScheduledEvent $*$, std::vector$<$ ScheduledEvent $*$ $>$, Event-Compare $>$ &)

    Returns true if this event is rescheduled on the specified queue, otherwise false.

Additional Inherited Members

5.100.1 Detailed Description

ScheduledEvent that executes repeatedly.

This will reschedule itself repeatedly at the appropriate interval.

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

- repast_hpc/Schedule.h
- repast_hpc/Schedule.cpp

5.101 repast::Request_Packet$<$ Content $>$ Class Template Reference

Contains information sent as agents are exchanged, either in response to requests or agent movement.

```
#include <RepastProcess.h>
```

Public Member Functions

- **Request_Packet** (std::vector$<$ Content $>$ $*$agentContent, std::map$<$ std::string, std::vector$<$ ProjectionInfo-Packet $*$ $>$ $>$ $*$projectionInfo)
- template$<$class Archive $>$
  void **serialize** (Archive &ar, const unsigned int version)

Public Attributes

- std::vector$<$ Content $>$ $*$ **agentContentPtr**
- std::map$<$ std::string,
  std::vector
  $<$ ProjectionInfoPacket $*$ $>$ $>$ $*$ **projectionInfoPtr**

Friends

- class **boost::serialization::access**

5.101.1 Detailed Description

template<typename Content>class repast::Request_Packet< Content >

Contains information sent as agents are exchanged, either in response to requests or agent movement.

Contains both agent raw information (of type 'Content') and projection information.

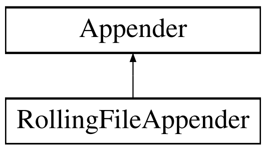
Note: A 'Packet' is responsible for deleting the objects to which it points This is essentially not optional: when boost sends the Packet via MPI the locations at which it places the different elements are not known (no 'new' is called in the user code). Some code must be written to track these down and delete, and it is manifestly easier to provide that code in the Packet itself than to rewrite where needed, inspecting the Packet for the locations

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

- repast_hpc/RepastProcess.h

5.102 RollingFileAppender Class Reference

Inheritance diagram for RollingFileAppender:



Public Member Functions

- **RollingFileAppender** (const string name, const string file_name, int max_backup, int max_size)
- virtual void **write** (const string &log_line)
- virtual void **close** ()

Additional Inherited Members

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

- repast_hpc/logger.cpp

5.103 repast::Schedule Class Reference

The simulation schedule queue.

`#include <Schedule.h>`

Public Types

- typedef boost::shared_ptr
  < Functor > FunctorPtr

  Typedef of for the functors that get scheduled.

Public Member Functions

- ScheduledEvent ∗ schedule_event (double at, FunctorPtr functor)

  Schedule the specified functor to execute once at the specified tick.
- ScheduledEvent ∗ schedule_event (double start, double interval, FunctorPtr func)

  Schedules the specified functor to execute start at start, and at the specified interval thereafter.
- void **execute** ()
- double getCurrentTick () const

  Gets the current simulation tick.
- double getNextTick () const

  Gets the next tick at which the next events will be executed.

5.103.1 Detailed Description

The simulation schedule queue.

This wraps a priority queue to schedule repast ScheduledEvents.

5.103.2 Member Function Documentation

5.103.2.1 double repast::Schedule::getCurrentTick ( ) const [inline]

Gets the current simulation tick.

Returns

the current simulation tick.

5.103.2.2 double repast::Schedule::getNextTick ( ) const [inline]

Gets the next tick at which the next events will be executed.

Returns

the next tick at which the next events will be executed.

5.103.2.3 ScheduledEvent ∗ repast::Schedule::schedule_event (double *at*, FunctorPtr *functor*)

Schedule the specified functor to execute once at the specified tick.

Parameters

at	the tick to execute at
functor	the functor to schedule

Returns

the event that has been scheduled

5.103.2.4 ScheduledEvent ∗ repast::Schedule::schedule_event (double *start*, double *interval*, FunctorPtr *func*)

Schedules the specified functor to execute start at start, and at the specified interval thereafter.

Parameters

start	
interval	
func	

Returns

the event that has been scheduled

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

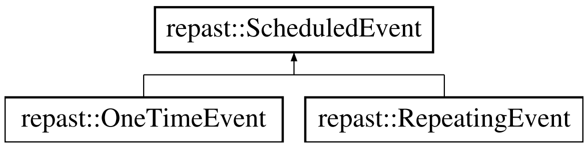
- repast_hpc/Schedule.h
- repast_hpc/Schedule.cpp

5.104 repast::ScheduledEvent Class Reference

The object that is placed (scheduled) in the priority queue for execution.

```
#include <Schedule.h>
```

Inheritance diagram for repast::ScheduledEvent:



Public Member Functions

- **ScheduledEvent** (double, RepastEvent ∗)
- virtual bool reschedule (std::priority_queue< ScheduledEvent ∗, std::vector< ScheduledEvent ∗ >, EventCompare > &)=0

  Returns true if this event is rescheduled on the specified queue, otherwise false.
- RepastEvent ∗ get_event ()

  Gets the RepastEvent that this ScheduleEvent wraps.

Protected Attributes

- RepastEvent ∗ **event**
- double **start**

Friends

- class **EventCompare**

5.104.1 Detailed Description

The object that is placed (scheduled) in the priority queue for execution.

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

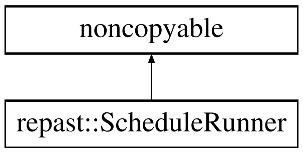
- repast_hpc/Schedule.h
- repast_hpc/Schedule.cpp

5.105 repast::ScheduleRunner Class Reference

Runs the Schedule by popping events off of the Schedule and executing them; also provides methods for scheduling events.

`#include <Schedule.h>`

Inheritance diagram for repast::ScheduleRunner:



Public Member Functions

- **ScheduleRunner** (boost::mpi::communicator ∗world)
- ScheduledEvent ∗ scheduleEvent (double at, Schedule::FunctorPtr func)

  Schedules the Functor to execute at the specified tick.
- ScheduledEvent ∗ scheduleEvent (double start, double interval, Schedule::FunctorPtr func)

  Schedules the Functor to execute at the specified start tick and every interval thereafter.
- void scheduleEndEvent (Schedule::FunctorPtr func)

  Schedules the specified functor to execute when the simulation ends.
- void scheduleStop (double at)

  Schedules the simulation to stop at the specified tick.
- void run ()

  Starts and runs the simulation schedule.
- double currentTick ()

  Gets the current tick.
- void stop ()

  Stops the simulation.
- const Schedule & schedule ()

  Gets the schedule executed by this simulation runner.

5.105.1 Detailed Description

Runs the Schedule by popping events off of the Schedule and executing them; also provides methods for scheduling events.

Simulation events should be scheduled for execution using this class which is accessible via RepastProcess::instance()->getScheduleRunner()

5.105.2 Member Function Documentation

5.105.2.1 double repast::ScheduleRunner::currentTick ( ) `[inline]`

Gets the current tick.

Returns

the current tick

5.105.2.2 const **Schedule**& repast::ScheduleRunner::schedule ( ) [inline]

Gets the schedule executed by this simulation runner.

Returns

the schedule used by this simulation runner.

5.105.2.3 void repast::ScheduleRunner::scheduleEndEvent (**Schedule::FunctorPtr** *func*)

Schedules the specified functor to execute when the simulation ends.

Parameters

func	the functor to execute when the simulatione ends

5.105.2.4 **ScheduledEvent** ∗ repast::ScheduleRunner::scheduleEvent (double *at,* **Schedule::FunctorPtr** *func*)

Schedules the Functor to execute at the specified tick.

Parameters

at	the time to execute at
func	the functor to execute

Returns

the event that was scheduled for the func

5.105.2.5 **ScheduledEvent** ∗ repast::ScheduleRunner::scheduleEvent (double *start,* double *interval,* **Schedule::FunctorPtr** *func*)

Schedules the Functor to execute at the specified start tick and every interval thereafter.

Parameters

start	the time to start at
interval	the interval to execute at
func	the functor to execute

Returns

the event that was scheduled for the func

5.105.2.6 void repast::ScheduleRunner::scheduleStop (double *at*)

Schedules the simulation to stop at the specified tick.

Parameters

at	the tick at which the simulation should stop

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

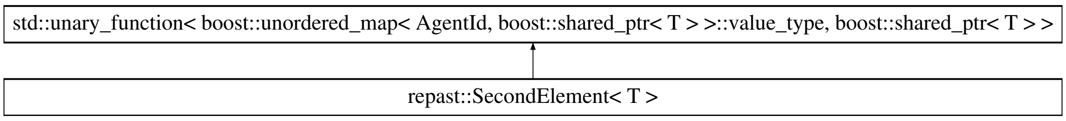
- repast_hpc/Schedule.h
- repast_hpc/Schedule.cpp

5.106 repast::SecondElement< T > Struct Template Reference

Unary function used in the transform_iterator that allows context iterators to return the agent maps values.

`#include <Context.h>`

Inheritance diagram for repast::SecondElement< T >:


std::unary_function< boost::unordered_map< AgentId, boost::shared_ptr< T > >::value_type, boost::shared_ptr< T > >
repast::SecondElement< T >

Public Member Functions

- boost::shared_ptr< T > **operator()** (const typename boost::unordered_map< AgentId, boost::shared_ptr< T > >::value_type &value) const

5.106.1 Detailed Description

template<typename T>struct repast::SecondElement< T >

Unary function used in the transform_iterator that allows context iterators to return the agent maps values.

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

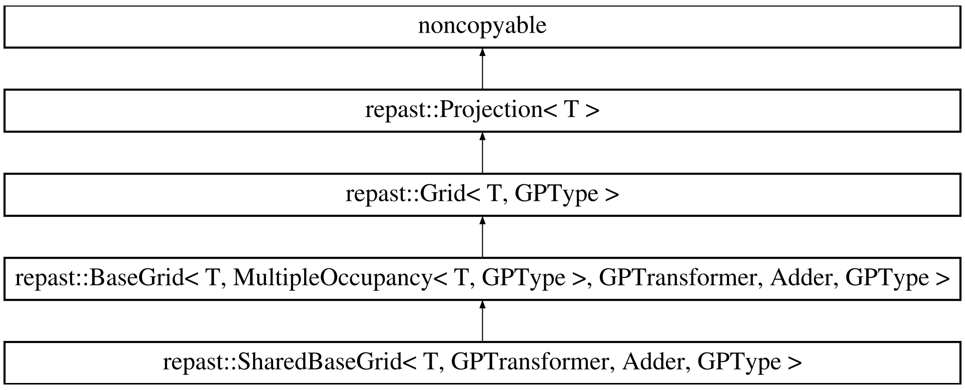
- repast_hpc/Context.h

5.107 repast::SharedBaseGrid< T, GPTransformer, Adder, GPType > Class Template Reference

Grid / Space implementation specialized for the distributed context.

`#include <SharedBaseGrid.h>`

Inheritance diagram for repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >:


noncopyable
repast::Projection< T >
repast::Grid< T, GPType >
repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >
repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >

Public Member Functions

- void **balance** ()

- SharedBaseGrid (std::string name, GridDimensions gridDims, std::vector< int > processDims, int buffer, boost::mpi::communicator ∗world)

    Creates a SharedGrid with the specified name.
- GridDimensions bounds () const

    Gets the local bounds of this SharedGrid.
- virtual const GridDimensions dimensions () const

    Gets the local bounds of this SharedGrid.
- void synchMove ()

    Synchronizes the movement of agents off on one grid and onto another.
- void initSynchBuffer (SharedContext< T > &context)

    Initializes the synch buffer operation.
- virtual bool moveTo (const AgentId &id, const std::vector< GPType > &newLocation)

    Moves the specified agent to the specified location.
- virtual bool moveTo (const AgentId &id, const Point< GPType > &pt)

    Moves the specified agent to the specified point.
- virtual void **removeAgent** (T ∗agent)
- virtual void **getRequiredAgents** (std::set< AgentId > &agentsToTest, std::set< AgentId > &agents-Required)
- virtual void getAgentsToPush (std::set< AgentId > &agentsToTest, std::map< int, std::set< AgentId > > &agentsToPush)

    Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.
- virtual void getInfoExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceiveFrom)

    Gets the set of processes with which this Projection exchanges projection info.
- virtual void getAgentStatusExchangePartners (std::set< int > &psToSendTo, std::set< int > &psToReceive-From)

    Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.
- virtual void **updateProjectionInfo** (ProjectionInfoPacket ∗pip, Context< T > ∗context)

Protected Types

- typedef repast::BaseGrid< T,
  MultipleOccupancy< T, GPType >
  , GPTransformer, Adder, GPType > **GridBaseType**

Protected Member Functions

- GridDimensions **createSendBufferBounds** (Neighbors::Location location)
- virtual void **synchMoveTo** (const AgentId &id, const Point< GPType > &pt)=0
- void **getMovingAgentInfo** (std::map< int, std::vector< AgentId > > agentsToMove, GridMovePackets< G-PType > outgoing)
- bool **locationIsInBuffer** (Point< GPType > pt)
- bool **agentIsInBuffer** (AgentId id)

Protected Attributes

- int **_buffer**
- GridDimensions **localBounds**
- Neighbors **nghs**
- std::vector< AgentId > **buffered**
- int **rank**
- boost::mpi::communicator ∗ **comm**

5.107.1 Detailed Description

template<typename T, typename GPTransformer, typename Adder, typename GPType>class repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >

Grid / Space implementation specialized for the distributed context.

Each SharedBaseGrid of the same name running on different processes is part of a pan process grid. This class manages this local part of the grid and its communication with its process neighbors. Users can specify a buffer size that determines how much of the neighboring grids are visible in this grid. For example, if this grid originates at 0x0 and ends at 3x3, a buffer of 1 would make the locations (4,0), (4,1) (4,2) ... (4,4) and (0,4), (1,4)... (4,4) visible in this grid. The SharedBaseGrid takes many template parameters. Default variations of these that define typical grids and spaces are given in SharedGrids in SharedSpace.h

Template Parameters

T	the type of objects contained by this BaseGrid
GPTransformer	transforms cell points according to the topology (e.g. periodic) of the BaseGrid.
Adder	determines how objects are added to the grid from its associated context.
GPType	the coordinate type of the grid point locations. This must be an int or a double.

5.107.2 Constructor & Destructor Documentation

5.107.2.1 template<typename T , typename GPTransformer , typename Adder , typename GPType > repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >::SharedBaseGrid (std::string *name*, GridDimensions *gridDims*, std::vector< int > *processDims*, int *buffer*, boost::mpi::communicator * *world*)

Creates a SharedGrid with the specified name.

Parameters

name	the name of this SharedBaseGrid
gridDims	the dimensions of the entire pan-process grid
processDims	the number of processes in each dimension. This must divide evenly into gridDims.
buffer	the size of the buffer between this part of the pan-process grid and its neighbors.

5.107.3 Member Function Documentation

5.107.3.1 template<typename T, typename GPTransformer, typename Adder, typename GPType> GridDimensions repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >::bounds ( ) const [inline]

Gets the local bounds of this SharedGrid.

The local bounds are the dimensions of the section of the pan-process grid represented by this SharedGrid.

Returns

the local bounds of this SharedGrid.

5.107.3.2 template<typename T, typename GPTransformer, typename Adder, typename GPType> virtual const GridDimensions repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >::dimensions ( ) const [inline], [virtual]

Gets the local bounds of this SharedGrid.

The local bounds are the dimensions of the section of the pan-process grid represented by this SharedGrid.

Returns

the local bounds of this SharedGrid.

Reimplemented from repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >.

5.107.3.3 template<typename T, typename GPTransformer, typename Adder, typename GPType> virtual void repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >::getAgentStatusExchangePartners (std::set< int > & *psToSendTo,* std::set< int > & *psToReceiveFrom*) [inline], [virtual]

Gets the set of processes with which this Projection exchanges agent status info- that is, the set of processes from which agents can move to this one or to which they can move when moving from this one.

In the most general case this will be all other processors. However, simulations where agents move in spaces will usually exchange agents only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implements repast::Grid< T, GPType >.

5.107.3.4 template<typename T , typename GPTransformer , typename Adder , typename GPType > void repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >::getAgentsToPush (std::set< AgentId > & *agentsToTest,* std::map< int, std::set< AgentId > > & *agentsToPush*) [virtual]

Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.

Generally spaces must push agents that are in 'buffer zones' and graphs must push local agents that are vertices to master edges where the other vertex is non- local. The results are returned per-process in the agentsToPush map.

Reimplemented from repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >.

Reimplemented in repast::SharedDiscreteSpace< T, GPTransformer, Adder >.

5.107.3.5 template<typename T, typename GPTransformer, typename Adder, typename GPType> virtual void repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >::getInfoExchangePartners (std::set< int > & *psToSendTo,* std::set< int > & *psToReceiveFrom*) [inline], [virtual]

Gets the set of processes with which this Projection exchanges projection info.

In the most general case this will be all other processors; this is the case for graphs, where agent connections can be arbitrary. However, spaces usually exchange information only with a small subset of 'neighbor' processes, which is knowable in advance and constant. To accommodate the general case, the algorithm for exchanging information must poll all other processes to see which are sending to this one; if this is known in advance, this additional (expensive) step can be skipped.

Implements repast::Grid< T, GPType >.

5.107.3.6 template<typename T, typename GPTransformer , typename Adder , typename GPType > void repast::SharedBaseGrid< T, GPTransformer, Adder, GPType >::initSynchBuffer (SharedContext< T > & *context*)

Initializes the synch buffer operation.

This should be called before synchronizing the buffers themselves.

Parameters

the	SharedContext that contains this SharedGrid projection.

5.107.3.7 template<typename T , typename GPTransformer , typename Adder , typename GPType> bool **repast::SharedBaseGrid**< T, GPTransformer, Adder, GPType >::moveTo (const **AgentId** & *id,* const std::vector< GPType > & *newLocation*) `[virtual]`

Moves the specified agent to the specified location.

Returns true if the move was successful otherwise false. The agent must be already added to the context associated with this space, otherwise this throws an out_of_range exception if the new location out of bounds.

Parameters

id	the id of the agent to move
newLocation	the location to move to

Returns

true if the move was successful, otherwise false

Reimplemented from repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >.

5.107.3.8 template<typename T , typename GPTransformer , typename Adder , typename GPType> bool **repast::SharedBaseGrid**< T, GPTransformer, Adder, GPType >::moveTo (const **AgentId** & *id,* const **Point**< GPType > & *pt*) `[virtual]`

Moves the specified agent to the specified point.

Parameters

id	the id of the agent to move
pt	where to move the agent to

Returns

true if the move was successful, otherwise false

Reimplemented from repast::BaseGrid< T, MultipleOccupancy< T, GPType >, GPTransformer, Adder, GPType >.

5.107.3.9 template<typename T , typename GPTransformer , typename Adder , typename GPType > void **repast::SharedBaseGrid**< T, GPTransformer, Adder, GPType >::synchMove ( )

Synchronizes the movement of agents off on one grid and onto another.

If there is any chance that an agent has moved off the local dimensions of this SharedGrid and into those managed by another then this must be called.

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

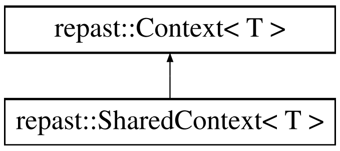
- repast_hpc/SharedBaseGrid.h

5.108 repast::SharedContext< T > Class Template Reference

Context implementation specialized for the parallel distributed simulation.

`#include <SharedContext.h>`

Inheritance diagram for repast::SharedContext< T >:



Public Types

- enum **filterLocalFlag** { **LOCAL** = 1, **NON_LOCAL** = 0 }
- typedef boost::filter_iterator
  < IsLocalAgent< T >, typename
  Context< T >::const_iterator > **const_local_iterator**
- typedef boost::filter_iterator
  < AgentStateFilter< T >
  , typename Context< T >
  ::const_iterator > **const_state_aware_iterator**
- typedef boost::filter_iterator
  < AgentStateFilter< T >
  , typename Context< T >
  ::const_bytype_iterator > **const_state_aware_bytype_iterator**
- typedef Projection< T >::RADIUS **RADIUS**

Public Member Functions

- **SharedContext** (boost::mpi::communicator ∗comm)
- const_local_iterator localBegin () const

  Gets the start of iterator over the local agents in this context.
- const_local_iterator localEnd () const

  Gets the end of an iterator over the local agents in this context.
- void removeAgent (const AgentId id)

  Removes the specified agent from this context.
- void removeAgent (T ∗agent)

  Removes the specified agent from this context.
- void importedAgentRemoved (const AgentId &id)

  Notifies this context that the specified non-local agent has been removed and this context should then delete that agent from itself.
- void incrementProjRefCount (const AgentId &id)

  Increments the projection reference count for the specified agent.
- void decrementProjRefCount (const AgentId &id)

  Decrements the projection reference count for the specified agent.
- const_state_aware_iterator begin (filterLocalFlag local)

  Gets the start of an iterator that will iterate over only local or non-local agents.
- const_state_aware_iterator end (filterLocalFlag local)

  Gets the end of an iterator that will iterate over only local or non-local agents.
- const_state_aware_bytype_iterator byTypeBegin (filterLocalFlag local, int type)

  Gets the start of an iterator that will iterate over only local or non-local agents of a certain type (per their AgentId value)
- const_state_aware_bytype_iterator byTypeEnd (filterLocalFlag local, int type)

  Gets the end of an iterator that will iterate over only local or non-local agents of a certain type (per their AgentId value)

- template< typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  SharedContext< T >
  ::const_state_aware_iterator > filteredBegin (filterLocalFlag local, filterStruct &fStruct)

  Gets the start of an iterator that will iterate over only local or non-local agents meeting the criteria of the user-defined struct (see IsLocalAgent for an example)
- template< typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  SharedContext< T >
  ::const_state_aware_iterator > filteredEnd (filterLocalFlag local, filterStruct &fStruct)

  Gets the end of an iterator that will iterate over only local or non-local agents meeting the criteria of the user-defined struct (see IsLocalAgent for an example)
- template< typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  SharedContext< T >
  ::const_state_aware_bytype_iterator > byTypeFilteredBegin (filterLocalFlag local, int type, filterStruct &fStruct)

  Gets the start of an iterator that will iterate over only local or non-local agents of the specified type and meeting the criteria of the user-defined struct (see IsLocalAgent for an example)
- template< typename filterStruct >
  boost::filter_iterator
  < filterStruct, typename
  SharedContext< T >
  ::const_state_aware_bytype_iterator > byTypeFilteredEnd (filterLocalFlag local, int type, filterStruct &fStruct)

  Gets the end of an iterator that will iterate over only local or non-local agents of the specified type and meeting the criteria of the user-defined struct (see IsLocalAgent for an example)
- void selectAgents (filterLocalFlag localOrNonLocalOnly, std::set< T ∗ > &selectedAgents, bool remove=false, int popSize=-1)

  Gets a set of pointers to all local or non-local agents in this context.
- void selectAgents (filterLocalFlag localOrNonLocalOnly, std::vector< T ∗ > &selectedAgents, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to all local or non-local agents in this context.
- void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::set< T ∗ > &selectedAgents, bool remove=false, int popSize=-1)

  Gets a set of pointers to a specified number of randomly selected local or non-local agents.
- void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::vector< T ∗ > &selectedAgents, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents.
- void selectAgents (filterLocalFlag localOrNonLocalOnly, std::set< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a set of pointers to all local or non-local agents in this context of a specified type (per their AgentId values).
- void selectAgents (filterLocalFlag localOrNonLocalOnly, std::vector< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to all local or non-local agents in this context of a specified type (per their AgentId values).
- void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::set< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a set of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values).
- void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::vector< T ∗ > &selectedAgents, int type, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values).

- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, std::set< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to all local or non-local agents in this context matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, std::vector< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to all local or non-local agents in this context matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::set< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to a specified number of randomly selected local or non-local agents matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::vector< T ∗ > &selectedAgents, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, std::set< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to all local or non-local agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, std::vector< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to all local or non-local agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::set< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a set of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values) and matching a user-defined filter.
- template<typename filterStruct >
  void selectAgents (filterLocalFlag localOrNonLocalOnly, int count, std::vector< T ∗ > &selectedAgents, int type, filterStruct &filter, bool remove=false, int popSize=-1)

  Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values) and matching a user-defined filter.
- bool keepsAgentsOnSyncProj ()

  Returns true if any of the projections in this context will try to 'keep' non-local agents during a synchronize projection operation.
- bool **sendsSecondaryDataOnStatusExchange** ()
- void **getProjInfoExchangePartners** (std::set< int > &sends, std::set< int > &recvs)
- void **getAgentStatusInfoExchangePartners** (std::set< int > &sends, std::set< int > &recvs)
- void getRequiredAgents (std::set< AgentId > &agentsToTest, std::set< AgentId > &agentsToKeep, RADIUS radius=Projection< T >::PRIMARY)

  Given a set of agents to test, returns the set of those agents that must be kept in order to keep required projection information.
- void getNonlocalAgentsToDrop (std::set< AgentId > &agentsToKeep, std::set< AgentId > &agentsToDrop, RADIUS radius=Projection< T >::PRIMARY)

  Given an initial set of agents that must be kept a priori, add any agents that must be kept due to projection requirements, and return the set of all non-local agents that can be dropped.
- void **getAgentsToPushToOtherProcesses** (std::map< int, std::set< AgentId > > &agentsToPush)
- virtual void addProjection (Projection< T > ∗projection)

  Adds the specified projection to this context.

Public Attributes

- IsLocalAgent< T > **localPredicate**
- AgentStateFilter< T > **LOCAL_FILTER**
- AgentStateFilter< T > **NON_LOCAL_FILTER**
- std::vector< std::string > **getAgentsToPushProjOrder**

Additional Inherited Members

5.108.1 Detailed Description

template<typename T>class repast::SharedContext< T >

Context implementation specialized for the parallel distributed simulation.

A SharedContext contains both local, that is, agents whose behavior is run on the SharedContext's process and foreign agents, that is, copies of agents whose behavior is run on some other process.

Parameters

T	the type of agents in the context.

5.108.2 Member Function Documentation

5.108.2.1 template<typename T > void repast::SharedContext< T >::addProjection (Projection< T > ∗ *projection*) [virtual]

Adds the specified projection to this context.

All the agents in this context will be added to the Projection. Any agents subsequently added to this context will also be added to the Projection.

Parameters

projection	the projection to add

Reimplemented from repast::Context< T >.

5.108.2.2 template<typename T > boost::filter_iterator< AgentStateFilter< T >, typename Context< T >::const_iterator > repast::SharedContext< T >::begin (filterLocalFlag *local*)

Gets the start of an iterator that will iterate over only local or non-local agents.

Parameters

local	flag indicating whether local or non-local agents are to be included

5.108.2.3 template<typename T > boost::filter_iterator< AgentStateFilter< T >, typename Context< T >::const_bytype_iterator > repast::SharedContext< T >::byTypeBegin (filterLocalFlag *local,* int *type*)

Gets the start of an iterator that will iterate over only local or non-local agents of a certain type (per their AgentId value)

Parameters

local	flag indicating whether local or non-local agents are to be included
type	type to included

5.108.2.4 template<typename T > boost::filter_iterator< **AgentStateFilter**< T >, typename **Context**< T >::const_bytype_iterator > **repast::SharedContext**< T >::byTypeEnd (filterLocalFlag *local,* int *type*)

Gets the end of an iterator that will iterate over only local or non-local agents of a certain type (per their AgentId value)

Parameters

local	flag indicating whether local or non-local agents are to be included
type	type to included

5.108.2.5 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename **SharedContext**< T >::const_state_aware_bytype_iterator > **repast::SharedContext**< T >::byTypeFilteredBegin (filterLocalFlag *local,* int *type,* filterStruct & *fStruct*)

Gets the start of an iterator that will iterate over only local or non-local agents of the specified type and meeting the criteria of the user-defined struct (see IsLocalAgent for an example)

Parameters

local	flag indicating whether local or non-local agents are to be included
type	type to be included
filter	struct with unary operator (boost::shared_ptr<T>) that returns true or false; used to selectively include agents.

Template Parameters

filterStruct	the class of the filter to be used

5.108.2.6 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename **SharedContext**< T >::const_state_aware_bytype_iterator > **repast::SharedContext**< T >::byTypeFilteredEnd (filterLocalFlag *local,* int *type,* filterStruct & *fStruct*)

Gets the end of an iterator that will iterate over only local or non-local agents of the specified type and meeting the criteria of the user-defined struct (see IsLocalAgent for an example)

Parameters

local	flag indicating whether local or non-local agents are to be included
type	type to be included
filter	struct with unary operator (boost::shared_ptr<T>) that returns true or false; used to selectively include agents.

Template Parameters

filterStruct	the class of the filter to be used

5.108.2.7 template<typename T > void **repast::SharedContext**< T >::decrementProjRefCount (const **AgentId** & *id*)

Decrements the projection reference count for the specified agent.

Parameters

id	the id of the agent

5.108.2.8 template<typename T > boost::filter_iterator< **AgentStateFilter**< T >, typename **Context**< T >::const_iterator > **repast::SharedContext**< T >::end (filterLocalFlag *local*)

Gets the end of an iterator that will iterate over only local or non-local agents.

Parameters

local	flag indicating whether local or non-local agents are to be included

5.108.2.9 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename **SharedContext**< T >::const_state_aware_iterator > **repast::SharedContext**< T >::filteredBegin (filterLocalFlag *local,* filterStruct & *fStruct*)

Gets the start of an iterator that will iterate over only local or non-local agents meeting the criteria of the user-defined struct (see IsLocalAgent for an example)

Parameters

local	flag indicating whether local or non-local agents are to be included
filter	struct with unary operator (boost::shared_ptr<T>) that returns true or false; used to selectively include agents.

Template Parameters

filterStruct	the class of the filter to be used

5.108.2.10 template<typename T > template<typename filterStruct > boost::filter_iterator< filterStruct, typename **SharedContext**< T >::const_state_aware_iterator > **repast::SharedContext**< T >::filteredEnd (filterLocalFlag *local,* filterStruct & *fStruct*)

Gets the end of an iterator that will iterate over only local or non-local agents meeting the criteria of the user-defined struct (see IsLocalAgent for an example)

Parameters

local	flag indicating whether local or non-local agents are to be included
filter	struct with unary operator (boost::shared_ptr<T>) that returns true or false; used to selectively include agents.

Template Parameters

filterStruct	the class of the filter to be used

5.108.2.11 template<typename T > void **repast::SharedContext**< T >::importedAgentRemoved (const **AgentId** & *id*)

Notifies this context that the specified non-local agent has been removed and this context should then delete that agent from itself.

Parameters

id	the id of the agent that was removed

5.108.2.12 template<typename T > void repast::SharedContext< T >::incrementProjRefCount (const AgentId & *id*)

Increments the projection reference count for the specified agent.

Parameters

id	the id of the agent

5.108.2.13 template<typename T > bool **repast::SharedContext**< T >::keepsAgentsOnSyncProj ( )

Returns true if any of the projections in this context will try to 'keep' non-local agents during a synchronize projection operation.

(Generally graphs keep local agents that are part of master copies of links, but spaces do not keep any local agents.)

5.108.2.14 template<typename T > boost::filter_iterator< **IsLocalAgent**< T >, typename **Context**< T >::const_iterator > **repast::SharedContext**< T >::localBegin ( ) const

Gets the start of iterator over the local agents in this context.

The iterator derefrences into shared_ptr<T>. The actual agent can be accessed by dereferencing the iter: (∗iter)->getId() for example.

Returns

the start of iterator over the local agents in this context.

5.108.2.15 template<typename T > boost::filter_iterator< **IsLocalAgent**< T >, typename **Context**< T >::const_iterator > **repast::SharedContext**< T >::localEnd ( ) const

Gets the end of an iterator over the local agents in this context.

The iterator derefrences into shared_ptr<T>. The actual agent can be accessed by derefrenceing the iter: (∗iter)->getId() for example.

5.108.2.16 template<typename T > void **repast::SharedContext**< T >::removeAgent (const **AgentId** *id*)

Removes the specified agent from this context.

If the agent is non-local, this checks to make sure that it is not referenced by any projection before its removed.

Parameters

id	the id of the agent to remove

5.108.2.17 template<typename T > void **repast::SharedContext**< T >::removeAgent (T ∗ *agent*)

Removes the specified agent from this context.

If the agent is non-local, this checks to make sure that it is not referenced by any projection before its removed.

Parameters

agent	the agent to remove

5.108.2.18 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly*, std::set< T ∗ > & *selectedAgents*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to all local or non-local agents in this context.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.19 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* std::vector< T ∗ > & *selectedAgents,* bool *remove* = `false`*,* int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to all local or non-local agents in this context.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.20 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::set< T ∗ > & *selectedAgents,* bool *remove* = `false`*,* int *popSize* = `-1`)

Gets a set of pointers to a specified number of randomly selected local or non-local agents.

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected

out	*selectedAgents*	a set into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.21 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::vector< T ∗ > & *selectedAgents,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents.

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
out	*selectedAgents*	a vector into which the pointers to the agents will be placed
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.22 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* std::set< T ∗ > & *selectedAgents,* int *type,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to all local or non-local agents in this context of a specified type (per their AgentId values).

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
out	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.23 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* std::vector< T ∗ > & *selectedAgents,* int *type,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to all local or non-local agents in this context of a specified type (per their AgentId values).

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.24 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::set< T * > & *selectedAgents,* int *type,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values).

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.25 template<typename T > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::vector< T * > & *selectedAgents,* int *type,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values).

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

5.108.2.26 template< typename T > template< typename filterStruct > void repast::SharedContext< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* std::set< T ∗ > & *selectedAgents,* filterStruct & *filter,* bool *remove =* `false`*,* int *popSize =* `-1`)

Gets a set of pointers to all local or non-local agents in this context matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.108.2.27 template< typename T > template< typename filterStruct > void repast::SharedContext< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* std::vector< T ∗ > & *selectedAgents,* filterStruct & *filter,* bool *remove =* `false`*,* int *popSize =* `-1`)

Gets a randomly ordered vector of pointers to all local or non-local agents in this context matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process

out	*selectedAgents*	a vector into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.108.2.28 template< typename T > template< typename filterStruct > void repast::SharedContext< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::set< T * > & *selectedAgents,* filterStruct & *filter,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to a specified number of randomly selected local or non-local agents matching a user-defined filter.

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
out	*selectedAgents*	a set into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.108.2.29 template< typename T > template< typename filterStruct > void repast::SharedContext< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::vector< T * > & *selectedAgents,* filterStruct & *filter,* bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents matching a user-defined filter.

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.108.2.30 template<typename T > template<typename filterStruct > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly*, std::set< T ∗ > & *selectedAgents*, int *type*, filterStruct & *filter*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a set of pointers to all local or non-local agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.108.2.31 template<typename T > template<typename filterStruct > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly*, std::vector< T ∗ > & *selectedAgents*, int *type*, filterStruct & *filter*, bool *remove* = `false`, int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to all local or non-local agents in this context of a specified type (per their AgentId values) and matching a user-defined filter.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.108.2.32 template<typename T > template<typename filterStruct > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::set< T * > & *selectedAgents,* int *type,* filterStruct & *filter,* bool *remove* = `false`*,* int *popSize* = `-1`)

Gets a set of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values) and matching a user-defined filter.

If the set passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original set will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a set into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

5.108.2.33 template<typename T > template<typename filterStruct > void **repast::SharedContext**< T >::selectAgents (filterLocalFlag *localOrNonLocalOnly,* int *count,* std::vector< T * > & *selectedAgents,* int *type,* filterStruct & *filter,* bool *remove* = `false`*,* int *popSize* = `-1`)

Gets a randomly ordered vector of pointers to a specified number of randomly selected local or non-local agents of a specified type (per their AgentId values) and matching a user-defined filter.

If the vector passed contains any elements when this method is called, the agents pointed to by those elements will be omitted from the selection.

If the 'remove' parameter is set to true, any elements in the original vector will be removed before the method returns.

The popSize parameter is used when the method is repeatedly called on a population whose size is known. Calls to this method typically begin by determining the size of the (valid) population to be sampled; if this is known, it can be provided here, improving performance.

Parameters

	localOrNon-LocalOnly	flag that indicates that the agents selected will be drawn only from agents either local or non-local to this process
	count	the number of agents to be selected. If this exceeds the number that can possibly be selected, all possible agents will be selected
`out`	*selectedAgents*	a vector into which the pointers to the agents will be placed
	type	numeric type of agent to be selected
	filter	user-defined filter specifying any criteria agents to be selected must meet
	remove	if true, remove any elements originally in the set before the set is returned (default is false)
	popSize	size of the population from which the sample will be drawn

Template Parameters

filterStruct	the type of the filter to be applied to the agents

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

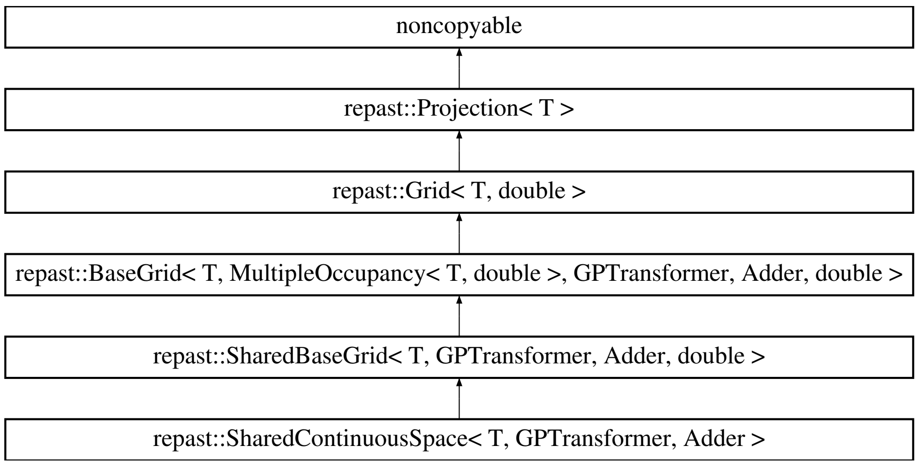
- repast_hpc/SharedContext.h

5.109 repast::SharedContinuousSpace< T, GPTransformer, Adder > Class Template Reference

Continuous space SharedBaseGrid implementation.

`#include <SharedContinuousSpace.h>`

Inheritance diagram for repast::SharedContinuousSpace< T, GPTransformer, Adder >:


noncopyable
repast::Projection< T >
repast::Grid< T, double >
repast::BaseGrid< T, MultipleOccupancy< T, double >, GPTransformer, Adder, double >
repast::SharedBaseGrid< T, GPTransformer, Adder, double >
repast::SharedContinuousSpace< T, GPTransformer, Adder >

Public Member Functions

- **SharedContinuousSpace** (std::string name, GridDimensions gridDims, std::vector< int > processDims, int buffer, boost::mpi::communicator ∗world)
- template<typename AgentContent , typename Provider , typename AgentsCreator >
  void synchBuffer (SharedContext< T > &context, Provider &provider, AgentsCreator &creator)

  Synchronize the buffer area of this SharedGrid with its neighbors.

- template<typename AgentContent , typename ContentProvider , typename ContentReceiver >
  void **synchBuffer** (SharedContext< T > &context, ContentProvider &provider, ContentReceiver &receiver)

Protected Member Functions

- virtual void **synchMoveTo** (const AgentId &id, const Point< double > &pt)

5.109.1 Detailed Description

template<typename T, typename GPTransformer, typename Adder>class repast::SharedContinuousSpace< T, GPTransformer, Adder >

Continuous space SharedBaseGrid implementation.

This primarily adds the buffer synchronization appropriate for this type. Default templated typical SharedContinuous-Spaces are defined in SharedGrids.

See Also

SharedBaseGrid for more details.

Template Parameters

T	the type of objects contained by this BaseGrid
GPTransformer	transforms cell points according to the topology (e.g. periodic) of the BaseGrid.
Adder	determines how objects are added to the grid from its associated context.

5.109.2 Member Function Documentation

5.109.2.1 template<typename T , typename GPTransformer , typename Adder > template<typename AgentContent , typename Provider , typename AgentsCreator > void repast::SharedContinuousSpace< T, GPTransformer, Adder >::synchBuffer (SharedContext< T > & *context,* Provider & *provider,* AgentsCreator & *creator*)

Synchronize the buffer area of this SharedGrid with its neighbors.

This will copy the buffer area from the neighbors into this SharedGrid. This should be called immediately after initSynchBuffer.

Parameters

context	the SharedContext that contains the agents in this SharedGrid.
provider	a class that provides AgentContent for the agents being buffered in neighboring grids.
creator	a class that creates a agents of type T when given AgentContent.

Template Parameters

T	the type of agent in this SharedGrid
AgentContent	the serializable struct or class that describes the state of agents.
Provider	a class that provides AgentContent for aagents, implementing void provide-Content(T∗ agent, std::vector<AgentContent>& out)
AgentsCreator	a class that creates agents given AgentContent, implementing void create-Agents(std::vector<AgentContent>& contents, std::vector<T∗>& out). Creating agents from the vector of content and placing them in out.

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

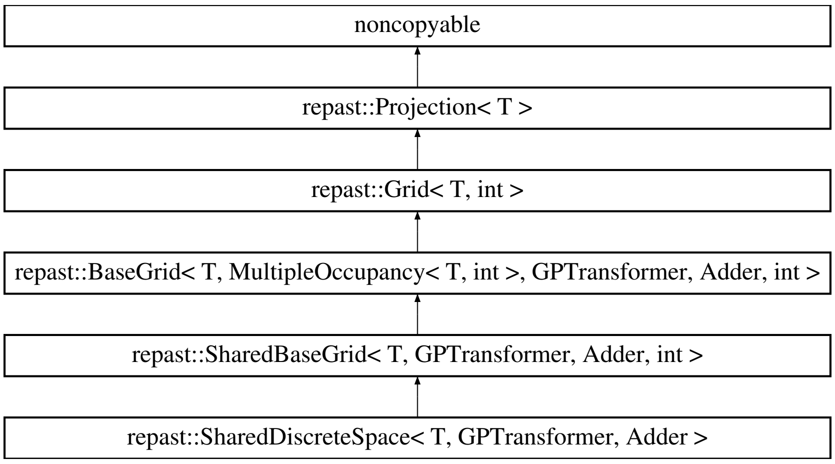
- repast_hpc/SharedContinuousSpace.h

5.110 repast::SharedDiscreteSpace< T, GPTransformer, Adder > Class Template Reference

Discrete matrix-like SharedBaseGrid implementation.

`#include <SharedDiscreteSpace.h>`

Inheritance diagram for repast::SharedDiscreteSpace< T, GPTransformer, Adder >:


noncopyable
repast::Projection< T >
repast::Grid< T, int >
repast::BaseGrid< T, MultipleOccupancy< T, int >, GPTransformer, Adder, int >
repast::SharedBaseGrid< T, GPTransformer, Adder, int >
repast::SharedDiscreteSpace< T, GPTransformer, Adder >

Public Member Functions

- **SharedDiscreteSpace** (std::string name, GridDimensions gridDims, std::vector< int > processDims, int buffer, boost::mpi::communicator ∗world)
- template<typename AgentContent , typename Provider , typename AgentsCreator >
  void synchBuffer (SharedContext< T > &context, Provider &provider, AgentsCreator &creator)

  Synchronize the buffer area of this SharedDiscreteSpace with its neighbors.
- virtual void getAgentsToPush (std::set< AgentId > &agentsToTest, std::map< int, std::set< AgentId > > &agentsToPush)

  Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.
- template<typename AgentContent , typename ContentProvider , typename ContentReceiver >
  void **synchBuffer** (SharedContext< T > &context, ContentProvider &provider, ContentReceiver &receiver)

Protected Member Functions

- virtual void **synchMoveTo** (const AgentId &id, const Point< int > &pt)

5.110.1 Detailed Description

template<typename T, typename GPTransformer, typename Adder>class repast::SharedDiscreteSpace< T, GPTransformer, Adder >

Discrete matrix-like SharedBaseGrid implementation.

This primarily adds the buffer synchronization appropriate for this type. Default templated typical SharedGrid types are defined in SharedGrids.

See Also

SharedBaseGrid for more details.

Template Parameters

T	the type of objects contained by this BaseGrid
GPTransformer	transforms cell points according to the topology (e.g. periodic) of the BaseGrid.
Adder	determines how objects are added to the grid from its associated context.

5.110.2 Member Function Documentation

5.110.2.1 template<typename T , typename GPTransformer , typename Adder > void **repast::SharedDiscreteSpace**< T, GPTransformer, Adder >::getAgentsToPush (std::set< **AgentId** > & *agentsToTest,* std::map< int, std::set< **AgentId** > > & *agentsToPush*) `[virtual]`

Given a set of agents, gets the agents that this projection implementation must 'push' to other processes.

Generally spaces must push agents that are in 'buffer zones' and graphs must push local agents that are vertices to master edges where the other vertex is non- local. The results are returned per-process in the agentsToPush map.

Reimplemented from repast::SharedBaseGrid< T, GPTransformer, Adder, int >.

5.110.2.2 template<typename T , typename GPTransformer , typename Adder > template<typename AgentContent , typename Provider , typename AgentsCreator > void **repast::SharedDiscreteSpace**< T, GPTransformer, Adder >::synchBuffer (**SharedContext**< T > & *context,* Provider & *provider,* AgentsCreator & *creator*)

Synchronize the buffer area of this SharedDiscreteSpace with its neighbors.

This will copy the buffer area from the neighbors into this SharedDiscreteSpace. This should be called immediately after initSynchBuffer.

Parameters

context	the SharedContext that contains the agents in this SharedDiscreteSpace.
provider	a class that provides AgentContent for the agents being buffered in neighboring grids.
creator	a class that creates a agents of type T when given AgentContent.

Template Parameters

T	the type of agent in this SharedDiscreteSpace
AgentContent	the serializable struct or class that describes the state of agents.
Provider	a class that provides AgentContent for aagents, implementing void provideContent(T∗ agent, std::vector<AgentContent>& out)
AgentsCreator	a class that creates agents given AgentContent, implementing void createAgents(std::vector<AgentContent>& contents, std::vector<T∗>& out). Creating agents from the vector of content and placing them in out.

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

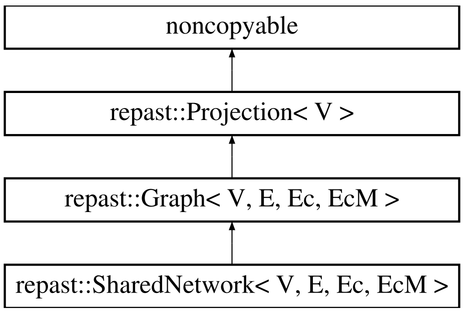
- repast_hpc/SharedDiscreteSpace.h

5.111 repast::SharedNetwork< V, E, Ec, EcM > Class Template Reference

Network implementation that can be shared across processes.

`#include <SharedNetwork.h>`

Inheritance diagram for repast::SharedNetwork< V, E, Ec, EcM >:


noncopyable
repast::Projection< V >
repast::Graph< V, E, Ec, EcM >
repast::SharedNetwork< V, E, Ec, EcM >

Public Member Functions

- SharedNetwork (std::string name, bool directed, EcM $*$edgeContentMgr)

  Creates a SharedNetwork with the specified name and whether or not the network is directed.
- void addSender (int rank)

  NON USER API.
- void removeSender (int rank)

  NON USER API Decrements the count of edges that are sent from rank to this network.
- void removeEdge (V $*$source, V $*$target)

  Removes the edge between source and target from this Graph.
- void addEdge (boost::shared_ptr$<$ E $>$ edge)

  Add an edge to this SharedNetwork.
- void synchRemovedEdges ()

  Synchronizes any removed edges that are have been copied across processes.
- virtual bool isMaster (E $*$e)

  Returns true if this is a master link; will be a master link if its master node is local.

Protected Member Functions

- virtual bool **addAgent** (boost::shared_ptr$<$ V $>$ agent)
- virtual void **removeAgent** (V $*$agent)
- virtual void **doAddEdge** (boost::shared_ptr$<$ E $>$ edge)

Friends

- template$<$typename Vertex , typename Edge , typename AgentContent , typename EdgeContent , typename EdgeManager , typename AgentCreator $>$
  void createComplementaryEdges (SharedNetwork$<$ Vertex, Edge, EdgeContent, EdgeManager $>$ $*$net, SharedContext$<$ Vertex $>$ &context, EdgeManager &edgeManager, AgentCreator &creator)

  Notifies other processes of any edges that have been created between nodes on this process and imported nodes.
- template$<$typename Vertex , typename Edge , typename EdgeContent , typename EdgeManager $>$
  void synchEdges (SharedNetwork$<$ Vertex, Edge, EdgeContent, EdgeManager $>$ $*$, EdgeManager &)

  Synchronize any edges that have been created as complementary edges.

Additional Inherited Members

5.111.1 Detailed Description

template$<$typename V, typename E, typename Ec, typename EcM$>$class repast::SharedNetwork$<$ V, E, Ec, EcM $>$

Network implementation that can be shared across processes.

Networks are shared across processes by creating edges between local and non-local agents on a process. The createComplementaryEdges function will create complementary edges across processes in those cases. For example, if an edge is created between A1 and B2 on process 1 where B2 is copy of B1 on process 2, then creating complementary edges will create a copy of that edge on process 2, importing A1 into process 2 if necessary.

Template Parameters

V	the agent (vertex) type
E	the edge type. The edge type must be contain a constructor that takes a source and target of type V and extends RepastEdge. RepastEdge can also be used.

5.111.2 Constructor & Destructor Documentation

5.111.2.1 template<typename V , typename E , typename Ec , typename EcM > repast::SharedNetwork< V, E, Ec, EcM >::SharedNetwork (std::string *name,* bool *directed,* EcM ∗ *edgeContentMgr*)

Creates a SharedNetwork with the specified name and whether or not the network is directed.

Parameters

the	network name
directed	if true the network will be directed, otherwise not.

5.111.3 Member Function Documentation

5.111.3.1 template<typename V , typename E , typename Ec , typename EcM > void repast::SharedNetwork< V, E, Ec, EcM >::addEdge (boost::shared_ptr< E > *edge*)

Add an edge to this SharedNetwork.

Parameters

edge	the edge to add

5.111.3.2 template<typename V , typename E , typename Ec , typename EcM > void repast::SharedNetwork< V, E, Ec, EcM >::addSender (int *rank*)

NON USER API.

Increments the count of edges that are sent from rank to this network.

5.111.3.3 template<typename V, typename E, typename Ec, typename EcM> virtual bool repast::SharedNetwork< V, E, Ec, EcM >::isMaster (E ∗ *e*) `[inline], [virtual]`

Returns true if this is a master link; will be a master link if its master node is local.

The master node is usually the edge 'source', but if the usesTargetAsMaster flag is set to true then the 'target' is the master node.

Implements repast::Graph< V, E, Ec, EcM >.

5.111.3.4 template<typename V , typename E , typename Ec , typename EcM > void repast::SharedNetwork< V, E, Ec, EcM >::removeEdge (V ∗ *source,* V ∗ *target*) `[virtual]`

Removes the edge between source and target from this Graph.

Parameters

source	the source of the edge
target	the target of the edge

Reimplemented from repast::Graph< V, E, Ec, EcM >.

5.111.4 Friends And Related Function Documentation

5.111.4.1 template<typename V, typename E, typename Ec, typename EcM> template<typename Vertex , typename Edge , typename AgentContent , typename EdgeContent , typename EdgeManager , typename AgentCreator > void createComplementaryEdges (SharedNetwork< Vertex, Edge, EdgeContent, EdgeManager > ∗ *net,* SharedContext< Vertex > & *context,* EdgeManager & *edgeManager,* AgentCreator & *creator*) [friend]

Notifies other processes of any edges that have been created between nodes on this process and imported nodes.

The other process will then create the complimentary edge. For example, if P1 creates an edge between A and B where B resides on P2, then this method will notify P2 to create the incoming edge A->B on its copy of B. Any unknown agents will be added to the context. For example, if P2 didn't have a reference to A, then A will be added to P2's context.

Parameters

net	the network in which to create the complementary edges or from which to send complementary edges
context	the context that contains the agents in the process
edgeManager	creates edges from EdgeContent and creates EdgeContent from an edge and a context.
creator	creates agents from AgentContent.

Template Parameters

Vertex	the vertex (agent) type
Edge	the edge type
AgentContent	the serializable struct or class that describes the agent state. It must contain a getId() method that returns the AgentId of the agent it describes.
EdgeContent	the serializable struct or class that describes edge state. At the very least EdgeContent must contain two public fields sourceContent and targetContent of type AgentContent. These represent the source and target of the edge.
EdgeManager	create edges from EdgeContent and provides EdgeContent given a context and an edge of type Edge. It must implement void provideEdgeContent(constEdge∗ edge, std::vector<EdgeContent>& edgeContent) and Edge∗ createEdge(repast::Context<Vertex>& context, EdgeContent& edge);
AgentCreator	creates agents from AgentContent, implementing the following method Vertex∗ createAgent(constAgentContent& content);

5.111.4.2 template<typename V, typename E, typename Ec, typename EcM> template<typename Vertex , typename Edge , typename EdgeContent , typename EdgeManager > void synchEdges (SharedNetwork< Vertex, Edge, EdgeContent, EdgeManager > ∗ *net,* EdgeManager & *edgeManager*) [friend]

Synchronizes any edges that have been created as complementary edges.

This only necessary if the edges have been deleted or their state has been changed in some way.

Parameters

net	the network in which to create the complementary edges or from which to send complementary edges

edgeManager	updates edges from EdgeContent and creates EdgeContent from an edge and a context.

Template Parameters

Vertex	the vertex (agent) type
Edge	the edge type
EdgeContent	the serializable struct or class that describes edge state.
EdgeManager	updates edges from EdgeContent and provides EdgeContent given a context and an edge of type Edge. It must implement void provideEdgeContent(const Edge∗ edge, std::vector<EdgeContent>& edgeContent) and void receiveEdgeContent(const EdgeContent& content);

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

- repast_hpc/SharedNetwork.h

5.112 repast::SharedSpaces< T > Struct Template Reference

Struct within which multiple kinds of shared space are typedef-ed.

`#include <SharedSpaces.h>`

Public Types

- typedef SharedDiscreteSpace< T, WrapAroundBorders, SimpleAdder< T > > SharedWrappedDiscreteSpace

  Discrete grid space with periodic (toroidal) borders.
- typedef SharedDiscreteSpace< T, StrictBorders, SimpleAdder< T > > SharedStrictDiscreteSpace

  Discrete grid space with strict borders.
- typedef SharedContinuousSpace< T, WrapAroundBorders, SimpleAdder< T > > SharedWrappedContinuousSpace

  Continuous space with periodic (toroidal) borders.
- typedef SharedContinuousSpace< T, StrictBorders, SimpleAdder< T > > SharedStrictContinuousSpace

  Continuous space with strict borders.

5.112.1 Detailed Description

template<typename T>struct repast::SharedSpaces< T >

Struct within which multiple kinds of shared space are typedef-ed.

5.112.2 Member Typedef Documentation

5.112.2.1 template<typename T > typedef **SharedContinuousSpace<T, StrictBorders, SimpleAdder<T> > repast::SharedSpaces< T >::SharedStrictContinuousSpace**

Continuous space with strict borders.

Any added agents are not given a location, but are in "grid limbo" until moved via a grid move call.

5.112.2.2 template< typename T > typedef SharedDiscreteSpace<T, StrictBorders, SimpleAdder<T> > repast::SharedSpaces< T >::SharedStrictDiscreteSpace

Discrete grid space with strict borders.

Any added agents are not given a location, but are in "grid limbo" until moved via a grid move call.

5.112.2.3 template< typename T > typedef SharedContinuousSpace<T, WrapAroundBorders, SimpleAdder<T> > repast::SharedSpaces< T >::SharedWrappedContinuousSpace

Continuous space with periodic (toroidal) borders.

Any added agents are not given a location, but are in "grid limbo" until moved via a grid move call.

5.112.2.4 template< typename T > typedef SharedDiscreteSpace<T, WrapAroundBorders, SimpleAdder<T> > repast::SharedSpaces< T >::SharedWrappedDiscreteSpace

Discrete grid space with periodic (toroidal) borders.

Any added agents are not given a location, but are in "grid limbo" until moved via a grid move call.

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

- repast_hpc/SharedSpaces.h

5.113 repast::SimpleAdder< T > Class Template Reference

Basic class for adding elements to grids.

`#include <GridComponents.h>`

Public Member Functions

- template< typename GridType >
  void **init** (GridDimensions dimensions, GridType ∗grid)
- bool **add** (boost::shared_ptr< T > agent)

5.113.1 Detailed Description

template< typename T > class repast::SimpleAdder< T >

Basic class for adding elements to grids.

NOTE: This does NOT actually add the element to the grid; this simply returns 'true' and leaves the actual addition to the grid up to the user. Other classes may do other things (e.g. add to a random location) but this one does NOT.

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

- repast_hpc/GridComponents.h

5.114 repast::SingleOccupancy< T, GPType > Class Template Reference

Single Occupancy cell accessor for accessing the occupants of locations in a Grid.

`#include <SingleOccupancy.h>`

Public Member Functions

- T ∗ get (const Point< GPType > &location) const

  Gets the object found at the specified location.
- void getAll (const Point< GPType > &location, std::vector< T ∗ > &out) const

  Gets the item found at the specified location.
- bool put (boost::shared_ptr< T > &agent, const Point< GPType > &location)

  Puts the specified item at the specified location.
- void remove (boost::shared_ptr< T > &agent, const Point< GPType > &location)

  Removes the specified item from the specified location.

5.114.1 Detailed Description

template<typename T, typename GPType>class repast::SingleOccupancy< T, GPType >

Single Occupancy cell accessor for accessing the occupants of locations in a Grid.

Each locations can have only a single occupant.

Parameters

T	the type of object in the Grid
GPType	the coordinate type of the grid point locations. This must be an int or a double.

5.114.2 Member Function Documentation

5.114.2.1 template<typename T , typename GPType > T ∗ repast::SingleOccupancy< T, GPType >::get (const Point< GPType > & *location*) const

Gets the object found at the specified location.

Parameters

location	the location to get the object at

Returns

the first object found at the specified location or 0 if there are no objects at the specified location.

5.114.2.2 template<typename T , typename GPType > void repast::SingleOccupancy< T, GPType >::getAll (const Point< GPType > & *location*, std::vector< T ∗ > & *out*) const

Gets the item found at the specified location.

Parameters

	location	the location to get the item at
out	*the*	found item will be returned in this vector

5.114.2.3 template<typename T , typename GPType > bool repast::SingleOccupancy< T, GPType >::put (boost::shared_ptr< T > & *agent*, const Point< GPType > & *location*)

Puts the specified item at the specified location.

Parameters

agent	the item to put
location	the location to put the item at

5.114.2.4 template< typename T , typename GPType > void **repast::SingleOccupancy**< T, GPType >::remove (boost::shared_ptr< T > & *agent*, const **Point**< GPType > & *location*)

Removes the specified item from the specified location.

Parameters

agent	the item to remove
location	the location to remove the item from

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

- repast_hpc/SingleOccupancy.h

5.115 repast::Spaces< T > Struct Template Reference

Struct within which multiple kinds of space are typedef-ed.

`#include <Spaces.h>`

Public Types

- typedef BaseGrid< T, SingleOccupancy< T, int >, StrictBorders, SimpleAdder< T >, int > **SingleStrictDiscreteSpace**
- typedef BaseGrid< T, SingleOccupancy< T, int >, WrapAroundBorders, SimpleAdder< T >, int > **SingleWrappedDiscreteSpace**
- typedef BaseGrid< T, MultipleOccupancy< T, int >, StrictBorders, SimpleAdder< T >, int > **MultipleStrictDiscreteSpace**
- typedef BaseGrid< T, MultipleOccupancy< T, int >, WrapAroundBorders, SimpleAdder< T >, int > **MultipleWrappedDiscreteSpace**
- typedef BaseGrid< T, SingleOccupancy< T, double >, StrictBorders, SimpleAdder< T >, double > **SingleStrictContinuousSpace**
- typedef BaseGrid< T, SingleOccupancy< T, double >, WrapAroundBorders, SimpleAdder< T >, double > **SingleWrappedContinuousSpace**
- typedef BaseGrid< T, MultipleOccupancy< T, double >, StrictBorders, SimpleAdder< T >, double > **MultipleStrictContinuousSpace**

- typedef BaseGrid< T,
  MultipleOccupancy< T, double >
  , WrapAroundBorders,
  SimpleAdder< T >, double > **MultipleWrappedContinuousSpace**

5.115.1 Detailed Description

template<typename T>struct repast::Spaces< T >

Struct within which multiple kinds of space are typedef-ed.

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

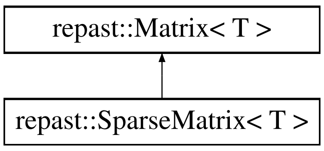
- repast_hpc/Spaces.h

5.116 repast::SparseMatrix< T > Class Template Reference

A sparse matrix implementation that stores values in a map.

`#include <matrix.h>`

Inheritance diagram for repast::SparseMatrix< T >:



Public Member Functions

- **SparseMatrix** (const SparseMatrix< T > &)
- SparseMatrix< T > & **operator=** (const SparseMatrix< T > &)
- SparseMatrix (const Point< int > &size, const T &defValue=T())

  Creates a DenseMatrix of the specified shape and default value.
- T & get (const Point< int > &index)

  Gets the value at the specified index.
- void set (const T &value, const Point< int > &index)

  Sets the value at the specified index.

Additional Inherited Members

5.116.1 Detailed Description

template<typename T>class repast::SparseMatrix< T >

A sparse matrix implementation that stores values in a map.

This should be used when the majority of the matrix cells contain the default value.

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

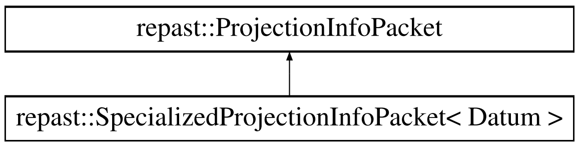
- repast_hpc/matrix.h

5.117 repast::SpecializedProjectionInfoPacket< Datum > Class Template Reference

Serializable packet that can contain projection information of a specific kind using the template parameter.

`#include <Projection.h>`

Inheritance diagram for repast::SpecializedProjectionInfoPacket< Datum >:


repast::ProjectionInfoPacket
repast::SpecializedProjectionInfoPacket< Datum >

Public Member Functions

- **SpecializedProjectionInfoPacket** (AgentId agentId)
- **SpecializedProjectionInfoPacket** (AgentId agentId, std::vector< Datum > projectionData)
- **SpecializedProjectionInfoPacket** (AgentId agentId, std::set< Datum > projectionData)
- template<class Archive >
  void **serialize** (Archive &ar, const unsigned int version)
- virtual bool **isEmpty** ()

Public Attributes

- std::vector< Datum > **data**

Friends

- class **boost::serialization::access**

5.117.1 Detailed Description

template<typename Datum>class repast::SpecializedProjectionInfoPacket< Datum >

Serializable packet that can contain projection information of a specific kind using the template parameter.

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

- repast_hpc/Projection.h

5.118 SRManager Class Reference

Coordinates send and receive between processes by notifying processes to expect a send from X other processes.

`#include <SRManager.h>`

Public Member Functions

- SRManager (boost::mpi::communicator *comm)

  Creates an SRManager that uses the specified communicator.
- SRManager (boost::mpi::communicator *comm, int *toSend, int *toRecv)

Creates an SRManager that uses the specified communicator, using the user-specified arrays instead of its internal arrays.

- void mark (int pos)

  Marks the position in the array as 'true' (sets to one).
- void setVal (int pos, int val)

  Sets the value at the given index in the array.
- void clear ()

  Clears the send and receive arrays (sets all values to 0).
- void retrieveSources ()

  Performs the actual send operation, populating the receive array with values from the other processes' send arrays.
- void retrieveSources (std::vector< int > &sources)

  Performs the send operation and populates the vector passed with values representing all elements in the array that have non-zero values after the receive.
- void retrieveSources (const std::vector< int > &targets, std::vector< int > &sources, int tag=0)

  Populates the send array based on the values listed in the 'targets' vector (which should be a list of process IDs to which this processer will send information) then performs the send operation, then populates the vector passed with values representing all elements in the receive array that have non-zero values after the receive.

5.118.1 Detailed Description

Coordinates send and receive between processes by notifying processes to expect a send from X other processes.

5.118.2 Constructor & Destructor Documentation

5.118.2.1 SRManager::SRManager (boost::mpi::communicator ∗ *comm*)

Creates an SRManager that uses the specified communicator.

Parameters

comm	the communicator to use

5.118.2.2 SRManager::SRManager (boost::mpi::communicator ∗ *comm,* int ∗ *toSend,* int ∗ *toRecv*)

Creates an SRManager that uses the specified communicator, using the user-specified arrays instead of its internal arrays.

The ability to use external arrays is a convenience for conditions in which it is useful to maintain the array of values for other purposes but exchange them using the SRManager.

Parameters

comm	the communicator to use
toSend	the array to be used as the send array
toRecv	the array to be used as the receive array If the pointer passed for the receive array is null, an internal array will be used. This is to provide for situations in which the user wishes to maintain the send array but not the receive array.

5.118.3 Member Function Documentation

5.118.3.1 void SRManager::mark (int *pos*)

Marks the position in the array as 'true' (sets to one).

Parameters

pos	the position in the array to be set, AKA the processor to which information will be sent.

5.118.3.2 void SRManager::retrieveSources (std::vector< int > & *sources*)

Performs the send operation and populates the vector passed with values representing all elements in the array that have non-zero values after the receive.

Parameters

sources	vector that will have a list of all processes that sent non-zero values to this one

5.118.3.3 void SRManager::retrieveSources (const std::vector< int > & *targets*, std::vector< int > & *sources*, int *tag* = 0)

Populates the send array based on the values listed in the 'targets' vector (which should be a list of process IDs to which this processer will send information) then performs the send operation, then populates the vector passed with values representing all elements in the receive array that have non-zero values after the receive.

Parameters

targets	vector of integers representing process to which this one intends to send information
sources	vector that will be populated with list of integers representing processes that will send this process information optional parameter, now obsolete (included for backward compatibility with boost-based SRManager prior to 2.0 release.

5.118.3.4 void SRManager::setVal (int *pos*, int *val*)

Sets the value at the given index in the array.

Note: Does not perform error checking; user should ensure that index value is valid.

Parameters

pos	index value for position in array to be set
val	value to which the array element should be set

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

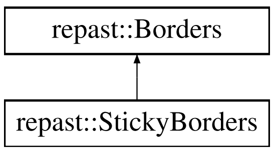
- repast_hpc/SRManager.h
- repast_hpc/SRManager.cpp

5.119 repast::StickyBorders Class Reference

Implements sticky border semantics: translates out side of the border are clamped to the border coordinates.

```
#include <GridComponents.h>
```

Inheritance diagram for repast::StickyBorders:



Public Member Functions

- **StickyBorders** (GridDimensions d)
- void **translate** (const std::vector< double > &oldPos, std::vector< double > &newPos, const std::vector< double > &displacement) const
- void **translate** (const std::vector< int > &oldPos, std::vector< int > &newPos, const std::vector< int > &displacement) const

Additional Inherited Members

5.119.1 Detailed Description

Implements sticky border semantics: translates out side of the border are clamped to the border coordinates.

Tranforms outside the border throw an exception.

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

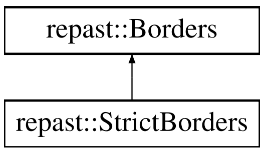
- repast_hpc/GridComponents.h
- repast_hpc/GridComponents.cpp

5.120 repast::StrictBorders Class Reference

Implements strict grid border semantics: anything outside the dimensions is out of bounds.

`#include <GridComponents.h>`

Inheritance diagram for repast::StrictBorders:



Public Member Functions

- **StrictBorders** (GridDimensions d)
- void **translate** (const std::vector< double > &oldPos, std::vector< double > &newPos, const std::vector< double > &displacement) const
- void **translate** (const std::vector< int > &oldPos, std::vector< int > &newPos, const std::vector< int > &displacement) const

Additional Inherited Members

5.120.1 Detailed Description

Implements strict grid border semantics: anything outside the dimensions is out of bounds.

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

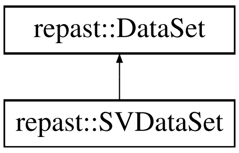
- repast_hpc/GridComponents.h
- repast_hpc/GridComponents.cpp

5.121 repast::SVDataSet Class Reference

Encapsulates data recording to a single plain text file, separating the recorded values using a specified separator value.

`#include <SVDataSet.h>`

Inheritance diagram for repast::SVDataSet:



Public Member Functions

- void record ()

  Records data from any added data sources.
- void write ()

  Writes any recorded data to a file.
- void close ()

  Closes the data set.

Friends

- class **SVDataSetBuilder**

5.121.1 Detailed Description

Encapsulates data recording to a single plain text file, separating the recorded values using a specified separator value.

An SVDataSet uses rank 0 to write to a single file from multiple pan-process data sources. A SVDataSet should be built using a SVDataSetBuilder.

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

- repast_hpc/SVDataSet.h
- repast_hpc/SVDataSet.cpp

5.122 repast::SVDataSetBuilder Class Reference

Used to build SVDataSets to record data in plain text tabular format.

`#include <SVDataSetBuilder.h>`

Public Member Functions

- SVDataSetBuilder (const std::string &file, const std::string &separator, const Schedule &schedule)

  Creates a SVDataSetBuilder that will create a SVDataSet that will write to the specified file and use the specified string as a data value separator.
- SVDataSetBuilder & addDataSource (SVDataSource ∗source)

Adds a DataSource to the DataSet produced by this builder.

- SVDataSet ∗ createDataSet ()

  Creates the DataSource defined by this builder.

5.122.1 Detailed Description

Used to build SVDataSets to record data in plain text tabular format.

Steps for use are:

1. Create a SVDataSetBuilder.
2. Add SVDataSources to the builder using the createSVDataSource functions. Each data source defines a column in the output and where the data for that column will be retrieved. Recording data on the SVDataSet produced by the builder will record this data for each column.
3. Call createDataSet to create the SVDataSet.
4. Schedule calls to record and write on the SVDataSet.

5.122.2 Constructor & Destructor Documentation

5.122.2.1 repast::SVDataSetBuilder::SVDataSetBuilder (const std::string & *file*, const std::string & *separator*, const Schedule & *schedule*)

Creates a SVDataSetBuilder that will create a SVDataSet that will write to the specified file and use the specified string as a data value separator.

Tick info will be gathered from the specified schedule.

Parameters

file	the file path where the data will be recorded to
separator	a string used to separate the data values (e.g. a ",").

5.122.3 Member Function Documentation

5.122.3.1 SVDataSetBuilder & repast::SVDataSetBuilder::addDataSource (SVDataSource ∗ *source*)

Adds a DataSource to the DataSet produced by this builder.

The createDataSource functions can be used to create Data Sources. Each data source defines a column in the output and where the data for that column will be retrieved. Recording data on the SVDataSet produced by the builder will record this data for each column.

Parameters

source	the data source to add

5.122.3.2 SVDataSet ∗ repast::SVDataSetBuilder::createDataSet ( )

Creates the DataSource defined by this builder.

This can only be called once. The caller is responsible for properly deleting the returned pointer.

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

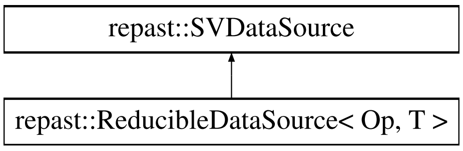
- repast_hpc/SVDataSetBuilder.h
- repast_hpc/SVDataSetBuilder.cpp

5.123 repast::SVDataSource Class Reference

Data source for data to be written into separated-value data sets.

```
#include <SVDataSource.h>
```

Inheritance diagram for repast::SVDataSource:



Public Types

- enum **DataType** { **INT**, **DOUBLE** }

Public Member Functions

- **SVDataSource** (const std::string &name)
- virtual void **record** ()=0
- virtual void **write** (Variable ∗var)=0
- virtual DataType **type** () const =0
- const std::string **name** () const

Protected Attributes

- std::string **_name**

5.123.1 Detailed Description

Data source for data to be written into separated-value data sets.

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

- repast_hpc/SVDataSource.h

5.124 repast::SyncStatus_Packet< Content > Class Template Reference

Class that contains information sent in conjunction with synchronizing agent status (agents moving or being removed from the simulation).

```
#include <RepastProcess.h>
```

Public Member Functions

- **SyncStatus_Packet** (std::vector< Content > ∗agentContent, std::map< std::string, std::vector< Projection-InfoPacket ∗ > > ∗projectionInfo, std::set< AgentId > ∗secondaryIds, AgentExporterInfo ∗exporterInfo)
- SyncStatus_Packet ∗ deleteExporterInfo ()

  This method includes a very odd construction that arises because the Packet must delete the exporter info on the process to which it has been sent, but it cannot delete the exporter info on the process from which it was sent.
- template<class Archive >
  void **serialize** (Archive &ar, const unsigned int version)

Public Attributes

- std::vector< Content > ∗ **agentContentPtr**
- std::map< std::string,
  std::vector
  < ProjectionInfoPacket ∗ > > ∗ **projectionInfoPtr**
- std::set< AgentId > ∗ **secondaryIdsPtr**
- AgentExporterInfo ∗ **exporterInfoPtr**

Friends

- class **boost::serialization::access**

5.124.1 Detailed Description

template<typename Content>class repast::SyncStatus_Packet< Content >

Class that contains information sent in conjunction with synchronizing agent status (agents moving or being removed from the simulation).

May contain secondary agent information (that is, agents that must newly be created as non-local agents on the receiving process due to obligations of projection contracts and the new existence of the agents being moved to that process).

Note the unusual requirement of the deletion of exporter information.

5.124.2 Member Function Documentation

5.124.2.1 template<typename Content> SyncStatus_Packet∗ repast::SyncStatus_Packet< Content >::deleteExporterInfo ( ) `[inline]`

This method includes a very odd construction that arises because the Packet *must* delete the exporter info on the process to which it has been sent, but it *cannot* delete the exporter info on the process from which it was sent.

The solution is to call this function manually on the receiving process, but not call it on the sending proc.

The pointer returned allows the abbreviation:

delete instance.deleteExporterInfo();

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

- repast_hpc/RepastProcess.h

5.125 repast::TDataSource< T > Class Template Reference

Interface for class that act as datasoures for DataSets.

`#include <TDataSource.h>`

Public Member Functions

- virtual T getData ()=0

  Gets the data.

5.125.1 Detailed Description

template< typename T >class repast::TDataSource< T >

Interface for class that act as datasoures for DataSets.

Template Parameters

T	the type of the data

5.125.2 Member Function Documentation

5.125.2.1 template<typename T> virtual T repast::TDataSource< T >::getData ( ) `[pure virtual]`

Gets the data.

Returns

the current data.

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

- repast_hpc/TDataSource.h

5.126 repast::Timer Class Reference

Simple timing class.

`#include <Utilities.h>`

Public Member Functions

- void start ()

  Starts the timer.
- long double stop ()

  Stops the timer and returns the number of milliseconds elapsed since calling start().

5.126.1 Detailed Description

Simple timing class.

Calling start() starts the timer, calling stop returns the milliseconds since calling start.

5.126.2 Member Function Documentation

5.126.2.1 long double repast::Timer::stop ( )

Stops the timer and returns the number of milliseconds elapsed since calling start().

Returns

the number of milliseconds elapsed since calling start().

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

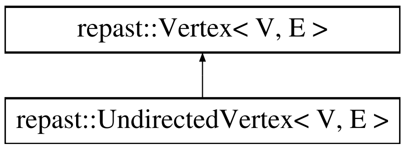
- repast_hpc/Utilities.h
- repast_hpc/Utilities.cpp

5.127 repast::UndirectedVertex< V, E > Class Template Reference

A vertex in an undirected network.

`#include <UndirectedVertex.h>`

Inheritance diagram for repast::UndirectedVertex< V, E >:



Public Member Functions

- **UndirectedVertex** (boost::shared_ptr< V > item)
- virtual boost::shared_ptr< E > removeEdge (Vertex< V, E > ∗other, EdgeType type)

  Removes the edge of the specified type between this Vertex and the specified Vertex.
- virtual boost::shared_ptr< E > findEdge (Vertex< V, E > ∗other, EdgeType type)

  Finds the edge of the specified type between this Vertex and the specified vertex.
- virtual void addEdge (Vertex< V, E > ∗other, boost::shared_ptr< E > edge, EdgeType type)

  Adds an edge of the specified type between this Vertex and the specified vertex.
- virtual void successors (std::vector< V ∗ > &out)

  Gets the successors of this Vertex.
- virtual void predecessors (std::vector< V ∗ > &out)

  Gets the predecessors of this Vertex.
- virtual void adjacent (std::vector< V ∗ > &out)

  Gets the Vertices adjacent to this Vertex.
- virtual void edges (EdgeType type, std::vector< boost::shared_ptr< E > > &out)

  Gets all the edges of the specified type in which this Vertex participates and return them in out.
- int inDegree ()

  Gets the in degree of this Vertex.
- int outDegree ()

  Gets the out degree of this Vertex.

5.127.1 Detailed Description

template<typename V, typename E>class repast::UndirectedVertex< V, E >

A vertex in an undirected network.

Template Parameters

V	the vertex type
E	the edge type. The edge type must be or extend RepastEdge.

5.127.2 Member Function Documentation

5.127.2.1 template<typename V , typename E > void repast::UndirectedVertex< V, E >::addEdge (Vertex< V, E > ∗ *other,* boost::shared_ptr< E > *edge,* EdgeType *type*) [virtual]

Adds an edge of the specified type between this Vertex and the specified vertex.

Parameters

edge	the edge to add
other	the other end of the edge
type	the type of edge to add

Implements repast::Vertex< V, E >.

5.127.2.2 template<typename V , typename E > void **repast::UndirectedVertex**< V, E >::adjacent (std::vector< V ∗ > & *out*) [virtual]

Gets the Vertices adjacent to this Vertex.

Parameters

out	*the*	vector where the adjacent vectors will be put

Implements repast::Vertex< V, E >.

5.127.2.3 template<typename V , typename E > void **repast::UndirectedVertex**< V, E >::edges (EdgeType *type,* std::vector< boost::shared_ptr< E > > & *out*) [virtual]

Gets all the edges of the specified type in which this Vertex participates and return them in out.

Parameters

	type	the type of edges to get
out	*where*	the edges will be put.

Implements repast::Vertex< V, E >.

5.127.2.4 template<typename V , typename E > boost::shared_ptr< E > **repast::UndirectedVertex**< V, E >::findEdge (**Vertex**< V, E > ∗ *other,* **EdgeType** *type*) [virtual]

Finds the edge of the specified type between this Vertex and the specified vertex.

Parameters

other	the other end of the edge
type	the type of edge to remove

Returns

the found edge, or 0.

Implements repast::Vertex< V, E >.

5.127.2.5 template<typename V , typename E > int **repast::UndirectedVertex**< V, E >::inDegree ( ) [virtual]

Gets the in degree of this Vertex.

Returns

the in degree of this Vertex.

Implements repast::Vertex< V, E >.

5.127.2.6 template<typename V , typename E > int **repast::UndirectedVertex**< V, E >::outDegree ( ) `[virtual]`

Gets the out degree of this Vertex.

Returns

the out degree of this Vertex.

Implements repast::Vertex< V, E >.

5.127.2.7 template<typename V , typename E > void **repast::UndirectedVertex**< V, E >::predecessors (std::vector< V ∗ > & *out*) `[virtual]`

Gets the predecessors of this Vertex.

Parameters

`out`	*the*	vector where any predecessors will be put

Implements repast::Vertex< V, E >.

5.127.2.8 template<typename V , typename E > boost::shared_ptr< E > **repast::UndirectedVertex**< V, E >::removeEdge (**Vertex**< V, E > ∗ *other*, **EdgeType** *type*) `[virtual]`

Removes the edge of the specified type between this Vertex and the specified Vertex.

Parameters

other	the other end of the edge
type	the type of edge to remove

Returns

the removed edge if such an edge was found, otherwise 0.

Implements repast::Vertex< V, E >.

5.127.2.9 template<typename V , typename E > void **repast::UndirectedVertex**< V, E >::successors (std::vector< V ∗ > & *out*) `[virtual]`

Gets the successors of this Vertex.

Parameters

`out`	*the*	vector where any successors will be put

Implements repast::Vertex< V, E >.

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

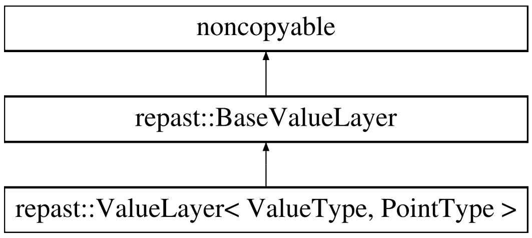
- repast_hpc/UndirectedVertex.h

5.128 repast::ValueLayer< ValueType, PointType > Class Template Reference

A collection that stores values at point locations.

```
#include <ValueLayer.h>
```

Inheritance diagram for repast::ValueLayer< ValueType, PointType >:



Public Member Functions

- **ValueLayer** (const std::string &name, const GridDimensions &dimensions)
- virtual ValueType & get (const Point< PointType > &pt)=0

  Gets the value at the specified point.
- virtual void set (const ValueType &value, const Point< PointType > &pt)=0

  Sets the value at the specified point.
- ValueType & operator[] (const Point< PointType > &pt)

  Gets the value at the specified point.
- const ValueType & operator[] (const Point< PointType > &pt) const

  Gets the value at the specified point.
- const GridDimensions dimensions () const

  Gets the dimensions of this ValueLayer.
- const Point< int > shape () const

  Gets the extents of this ValueLayer.

Protected Member Functions

- void translate (std::vector< PointType > &pt)

  Translates pt by dimensions origin.

Protected Attributes

- GridDimensions **_dimensions**

5.128.1 Detailed Description

template<typename ValueType, typename PointType>class repast::ValueLayer< ValueType, PointType >

A collection that stores values at point locations.

Template Parameters

ValueType	the type stored by the value layer.
the	coordinate type (int or double) of the point locations.

5.128.2 Member Function Documentation

5.128.2.1 template<typename ValueType, typename PointType> const GridDimensions repast::ValueLayer< ValueType, PointType >::dimensions ( ) const [inline]

Gets the dimensions of this ValueLayer.

Returns

the dimensions of this ValueLayer.

5.128.2.2 template<typename ValueType, typename PointType> virtual ValueType& repast::ValueLayer< ValueType, PointType >::get (const Point< PointType > & *pt*) [pure virtual]

Gets the value at the specified point.

If no value has been set at the specified point then this returns some default value. Subclasses will determine the default value.

param pt the location to get the value of

Returns

the value at the specified point, or if no value has been set, then some default value.

Implemented in repast::ContinuousValueLayer< ValueType, Borders >, and repast::DiscreteValueLayer< ValueType, Borders >.

5.128.2.3 template<typename ValueType , typename PointType> ValueType & repast::ValueLayer< ValueType, PointType >::operator[] (const Point< PointType > & *pt*)

Gets the value at the specified point.

If no value has been set at the specified point then this returns some default value. Subclasses will determine the default value.

param pt the location to get the value of

Returns

the value at the specified point, or if no value has been set, then some default value.

5.128.2.4 template<typename ValueType , typename PointType> const ValueType & repast::ValueLayer< ValueType, PointType >::operator[] (const Point< PointType > & *pt*) const

Gets the value at the specified point.

If no value has been set at the specified point then this returns some default value. Subclasses will determine the default value.

param pt the location to get the value of

Returns

the value at the specified point, or if no value has been set, then some default value.

5.128.2.5 template<typename ValueType, typename PointType> virtual void repast::ValueLayer< ValueType, PointType >::set (const ValueType & *value*, const Point< PointType > & *pt*) [pure virtual]

Sets the value at the specified point.

Parameters

value	the value
pt	the point where the value should be stored

Implemented in repast::ContinuousValueLayer< ValueType, Borders >, and repast::DiscreteValueLayer< ValueType, Borders >.

5.128.2.6 template<typename ValueType, typename PointType> const Point<int> repast::ValueLayer< ValueType, PointType >::shape ( ) const `[inline]`

Gets the extents of this ValueLayer.

Returns

the extents of this ValueLayer.

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

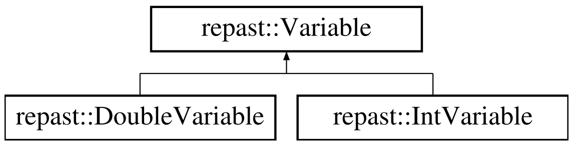
- repast_hpc/ValueLayer.h

5.129 repast::Variable Class Reference

Used in SVDataSet to manage and store the data.

`#include <Variable.h>`

Inheritance diagram for repast::Variable:



Public Member Functions

- virtual void write (size_t index, std::ofstream &out)=0
  Writes the data at the specified index to the specified ofstream.
- virtual void insert (double ∗array, size_t size)=0
  Inserts all the doubles in the double array into the collection of data stored in this Variable.
- virtual void insert (int ∗array, size_t size)=0
  Inserts all the ints in the int array into the collection of data stored in this Variable.
- virtual void clear ()=0
  Clears this Variable of all the data stored in it.

5.129.1 Detailed Description

Used in SVDataSet to manage and store the data.

5.129.2 Member Function Documentation

5.129.2.1 virtual void repast::Variable::insert (double ∗ *array*, size_t *size*) `[pure virtual]`

Inserts all the doubles in the double array into the collection of data stored in this Variable.

Parameters

array	the array to insert
size	the size of the array

Implemented in repast::DoubleVariable, and repast::IntVariable.

5.129.2.2 virtual void repast::Variable::insert (int ∗ *array,* size_t *size*) [pure virtual]

Inserts all the ints in the int array into the collection of data stored in this Variable.

Parameters

array	the array to insert
size	the size of the array

Implemented in repast::DoubleVariable, and repast::IntVariable.

5.129.2.3 virtual void repast::Variable::write (size_t *index,* std::ofstream & *out*) [pure virtual]

Writes the data at the specified index to the specified ofstream.

Parameters

index	the index of the data to write
out	the ofstream to write the data to

Implemented in repast::DoubleVariable, and repast::IntVariable.

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

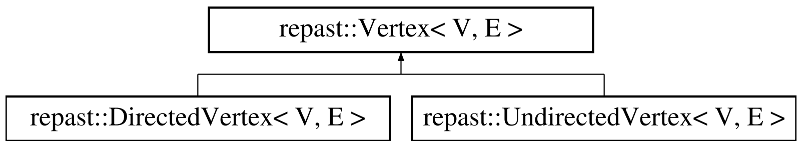
- repast_hpc/Variable.h

5.130 repast::Vertex< V, E > Class Template Reference

Used internally by repast graphs / networks to encapsulate Vertices.

`#include <Vertex.h>`

Inheritance diagram for repast::Vertex< V, E >:


repast::Vertex< V, E >
repast::DirectedVertex< V, E >
repast::UndirectedVertex< V, E >

Public Types

- enum EdgeType { **INCOMING**, **OUTGOING** }

  Enum the identifies whether an edge is incoming or outgoing.
- typedef boost::unordered_map
  < Vertex< V, E >
  ∗, boost::shared_ptr< E >
  , HashVertex< V, E > > AdjListMap

  Typedef for the adjacency list map that contains the other Vertices that this Vertex links to.
- typedef AdjListMap::iterator **AdjListMapIterator**

Public Member Functions

- Vertex (boost::shared_ptr< V > item)

  Creates a Vertex that contains the specified item.
- virtual boost::shared_ptr< E > removeEdge (Vertex ∗other, EdgeType type)=0

  Removes the edge of the specified type between this Vertex and the specified Vertex.
- virtual boost::shared_ptr< E > findEdge (Vertex ∗other, EdgeType type)=0

  Finds the edge of the specified type between this Vertex and the specified vertex.
- virtual void addEdge (Vertex< V, E > ∗other, boost::shared_ptr< E > edge, EdgeType type)=0

  Adds an edge of the specified type between this Vertex and the specified vertex.
- virtual void successors (std::vector< V ∗ > &out)=0

  Gets the successors of this Vertex.
- virtual void predecessors (std::vector< V ∗ > &out)=0

  Gets the predecessors of this Vertex.
- virtual void adjacent (std::vector< V ∗ > &out)=0

  Gets the Vertices adjacent to this Vertex.
- virtual void edges (EdgeType type, std::vector< boost::shared_ptr< E > > &out)=0

  Gets all the edges of the specified type in which this Vertex participates and return them in out.
- virtual int inDegree ()=0

  Gets the in degree of this Vertex.
- virtual int outDegree ()=0

  Gets the out degree of this Vertex.
- boost::shared_ptr< V > item () const

  Gets the item that this Vertex contains.

Protected Member Functions

- boost::shared_ptr< E > **removeEdge** (Vertex< V, E > ∗other, AdjListMap ∗adjMap)
- void **getItems** (AdjListMap ∗adjMap, std::vector< V ∗ > &out)
- void **edges** (AdjListMap ∗adjMap, std::vector< boost::shared_ptr< E > > &out)

Protected Attributes

- boost::shared_ptr< V > **ptr**

Friends

- struct **NodeGetter< V, E >**

5.130.1 Detailed Description

template<typename V, typename E>class repast::Vertex< V, E >

Used internally by repast graphs / networks to encapsulate Vertices.

Template Parameters

V	the type of object stored by in a Vertex.

E	the edge type of the network.

5.130.2 Constructor & Destructor Documentation

5.130.2.1 template<typename V, typename E > **repast::Vertex**< V, E >::Vertex (boost::shared_ptr< V > *item*)

Creates a Vertex that contains the specified item.

Parameters

item	the item the Vertex should contain

5.130.3 Member Function Documentation

5.130.3.1 template<typename V, typename E> virtual void **repast::Vertex**< V, E >::addEdge (**Vertex**< V, E > ∗ *other*, boost::shared_ptr< E > *edge*, **EdgeType** *type*) `[pure virtual]`

Adds an edge of the specified type between this Vertex and the specified vertex.

Parameters

edge	the edge to add
other	the other end of the edge
type	the type of edge to add

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.2 template<typename V, typename E> virtual void **repast::Vertex**< V, E >::adjacent (std::vector< V ∗ > & *out*) `[pure virtual]`

Gets the Vertices adjacent to this Vertex.

Parameters

`out`	*the*	vector where the adjacent vectors will be put

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.3 template<typename V, typename E> virtual void **repast::Vertex**< V, E >::edges (**EdgeType** *type*, std::vector< boost::shared_ptr< E > > & *out*) `[pure virtual]`

Gets all the edges of the specified type in which this Vertex participates and return them in out.

Parameters

	type	the type of edges to get
`out`	*where*	the edges will be put.

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.4 template<typename V, typename E> virtual boost::shared_ptr<E> **repast::Vertex**< V, E >::findEdge (**Vertex**< V, E > ∗ *other*, **EdgeType** *type*) `[pure virtual]`

Finds the edge of the specified type between this Vertex and the specified vertex.

Parameters

other	the other end of the edge
type	the type of edge to remove

Returns

the found edge, or 0.

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.5 template<typename V, typename E> virtual int **repast::Vertex**< V, E >::inDegree ( ) `[pure virtual]`

Gets the in degree of this Vertex.

Returns

the in degree of this Vertex.

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.6 template<typename V, typename E> boost::shared_ptr<V> **repast::Vertex**< V, E >::item ( ) const `[inline]`

Gets the item that this Vertex contains.

Returns

the item.

5.130.3.7 template<typename V, typename E> virtual int **repast::Vertex**< V, E >::outDegree ( ) `[pure virtual]`

Gets the out degree of this Vertex.

Returns

the out degree of this Vertex.

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.8 template<typename V, typename E> virtual void **repast::Vertex**< V, E >::predecessors (std::vector< V * > & *out*) `[pure virtual]`

Gets the predecessors of this Vertex.

Parameters

`out`	*the*	vector where any predecessors will be put

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.9 template<typename V, typename E> virtual boost::shared_ptr<E> **repast::Vertex**< V, E >::removeEdge (**Vertex**< V, E > * *other*, **EdgeType** *type*) `[pure virtual]`

Removes the edge of the specified type between this Vertex and the specified Vertex.

Parameters

other	the other end of the edge
type	the type of edge to remove

Returns

the removed edge if such an edge was found, otherwise 0.

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

5.130.3.10 template<typename V, typename E> virtual void **repast::Vertex**< V, E >::successors (std::vector< V ∗ > & *out*) [pure virtual]

Gets the successors of this Vertex.

Parameters

`out`	*the*	vector where any successors will be put

Implemented in repast::DirectedVertex< V, E >, and repast::UndirectedVertex< V, E >.

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

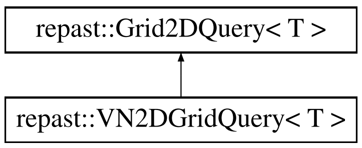
- repast_hpc/Vertex.h

5.131 repast::VN2DGridQuery< T > Class Template Reference

Neighborhood query that gathers neighbors in a Von Neumann (N, S, E, W) neighborhood.

`#include <VN2DGridQuery.h>`

Inheritance diagram for repast::VN2DGridQuery< T >:



Public Member Functions

- **VN2DGridQuery** (const Grid< T, int > ∗grid)
- virtual void query (const Point< int > &center, int range, bool includeCenter, std::vector< T ∗ > &out) const
  Queries the Grid for the Von Neumann neighbors surrounding the center point within a specified range.

Additional Inherited Members

5.131.1 Detailed Description

template<typename T>class repast::VN2DGridQuery< T >

Neighborhood query that gathers neighbors in a Von Neumann (N, S, E, W) neighborhood.

Template Parameters

T	the type of agents in the Grid

5.131.2 Member Function Documentation

5.131.2.1 template<typename T > void **repast::VN2DGridQuery**< T >::query (const **Point**< int > & *center,* int *range,* bool *includeCenter,* std::vector< T ∗ > & *out*) const [virtual]

Queries the Grid for the Von Neumann neighbors surrounding the center point within a specified range.

Parameters

	center	the center of the neighborhood
	range	the range of the neighborhood out from the center
	includeCenter	whether or not to include any agents at the center
out	*the*	neighboring agents will be returned in this vector

Implements repast::Grid2DQuery< T >.

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

- repast_hpc/VN2DGridQuery.h

5.132 repast::WrapAroundBorders Class Reference

Implements periodic wrap around style border semantics.

```
#include <GridComponents.h>
```

Public Member Functions

- **WrapAroundBorders** (GridDimensions dimensions)
- void **transform** (const std::vector< int > &in, std::vector< int > &out) const
- void **transform** (const std::vector< double > &in, std::vector< double > &out) const
- void **translate** (const std::vector< double > &oldPos, std::vector< double > &newPos, const std::vector< double > &displacement) const
- void **translate** (const std::vector< int > &oldPos, std::vector< int > &newPos, const std::vector< int > &displacement) const
- void **init** (const GridDimensions &dimensions)
- bool **isPeriodic** () const

5.132.1 Detailed Description

Implements periodic wrap around style border semantics.

Points that are outside the borders are wrapped until the point is inside the borders.

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

- repast_hpc/GridComponents.h
- repast_hpc/GridComponents.cpp