Stratmas Reference Manual

Generated by Doxygen 1.3.4

Wed Jan 31 14:27:32 2007

Contents

Chapter 1

Stratmas Hierarchical Index

1.1 Stratmas Class Hierarchy

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

Chapter 2

Stratmas Class Index

2.1 Stratmas Class List

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

Chapter 3

Stratmas File Index

3.1 Stratmas File List

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

Chapter 4

Stratmas Class Documentation

4.1 AccessRightHandler Class Reference

This class keeps track of which types of objects that the server may not change during a simulation.

`#include <AccessRightHandler.h>`

Static Public Member Functions

- bool **changeable** (const **Reference** &ref)

  Checks if the object with the provided **Reference**(p. 378) *is changeable or not.*

Static Private Attributes

- std::set< const **Type** *, **lessTypeP** > **sUnchangableTypes**

  The set of Types that may not change during a simulation.

4.1.1 Detailed Description

This class keeps track of which types of objects that the server may not change during a simulation.

Author:
Per Alexius

Date **Date**
2006/03/02 17:06:50

4.1.2 Member Function Documentation

4.1.2.1 bool AccessRightHandler::changeable (const Reference & *ref*) [static]

Checks if the object with the provided **Reference**(p. 378) is changeable or not.

Parameters:

ref The **Reference**(p. 378) to the object to check.

Returns:

True if the object with the provided **Reference**(p. 378) is changeable, false otherwise.

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

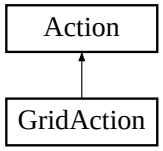
- AccessRightHandler.h
- AccessRightHandler.cpp

4.2 Action Class Reference

Super class for all actions.

`#include <Action.h>`

Inheritance diagram for Action::



Public Member Functions

- **Action** (**Grid** &target)
  Constructor.
- virtual ~**Action** ()
  Destructor.
- virtual void **carryOut** ()
  Carries out this action.

Protected Attributes

- **Grid** & **mTarget**
  The target of this action.

4.2.1 Detailed Description

Super class for all actions.

Author:
Per Alexius

Date **Date**
2006/03/06 12:55:07

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Action::Action (Grid & *target*) `[inline]`

Constructor.

Parameters:
target The target of this Action.

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

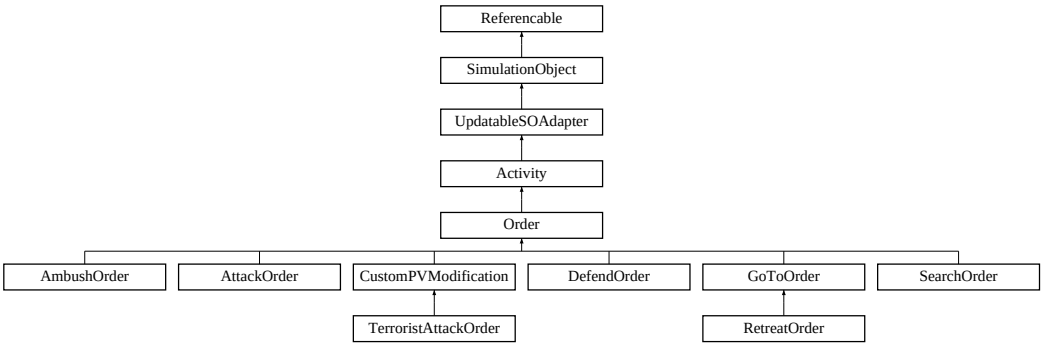
- Action.h

4.3 Activity Class Reference

Abstract super class for all activities.

`#include <Activity.h>`

Inheritance diagram for Activity::



Public Member Functions

- **Activity** ()

  Default Constructor.

- **Activity** (const **DataObject** &d)

  Creates an Activity from the provided **DataObject**(p. 145).

- virtual ~**Activity** ()

  Destructor.

- virtual void **prepareForSimulation** (**Grid** &g, **Time** currentTime)

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- virtual void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- virtual void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- virtual void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- **Time startTime** () const

  Accessor for the start time.

- virtual **Shape** ∗ **location** () const =0

  Accessor for the area.

- virtual bool **isActive** (**Time** t)=0

Checks if this activity is active at time t.

- virtual void **perform** (**Element** *e, double fraction=1.0)=0

  Performs this Activity.

Protected Attributes

- bool **mActive**

  Indicates if this activity is currently executed.

- **Time mStart**

  The start time.

4.3.1 Detailed Description

Abstract super class for all activities.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Activity::Activity (const DataObject & *d*)

Creates an Activity from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.3.3 Member Function Documentation

4.3.3.1 void Activity::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

Reimplemented in **Order** (p. 310), **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 514), **DefendOrder** (p. 166), and **AmbushOrder** (p. 43).

4.3.3.2 virtual bool Activity::isActive (Time *t*) [pure virtual]

Checks if this activity is active at time t.

Parameters:
t The time for which to check.

Returns:
True if this activity is active at the specified time.

Implemented in **Order** (p. 310), **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 166), **AmbushOrder** (p. 43), and **SearchOrder** (p. 402).

4.3.3.3 virtual Shape* Activity::location () const [pure virtual]

Accessor for the area.

Returns:
The area or null if this activity does not have an area.

Implemented in **Order** (p. 311).

4.3.3.4 void Activity::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **UpdatableSOAdapter** (p. 565).

Reimplemented in **Order** (p. 311), **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 167), and **AmbushOrder** (p. 43).

4.3.3.5 virtual void Activity::perform (Element * *e*, double *fraction* = 1.0) [pure virtual]

Performs this Activity.

Parameters:
e The **Element**(p. 180) that should perform this Activity.

fraction The fraction of the performers total capacity that this activity is performed with.

Implemented in **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 515), **AttackOrder** (p. 56), **DefendOrder** (p. 167), **AmbushOrder** (p. 44), **GoToOrder** (p. 226), **RetreatOrder** (p. 395), and **SearchOrder** (p. 402).

4.3.3.6 void Activity::prepareForSimulation (Grid & *g*, Time *currentTime*) [virtual]

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Parameters:

g The **Grid**(p. 227).

currentTime The current simulation time.

Reimplemented in **CustomPVModification** (p. 143).

4.3.3.7 void Activity::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

Reimplemented in **Order** (p. 312), **CustomPVModification** (p. 143), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 167), and **AmbushOrder** (p. 44).

4.3.3.8 Time Activity::startTime () const [inline]

Accessor for the start time.

Returns:

The start time.

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

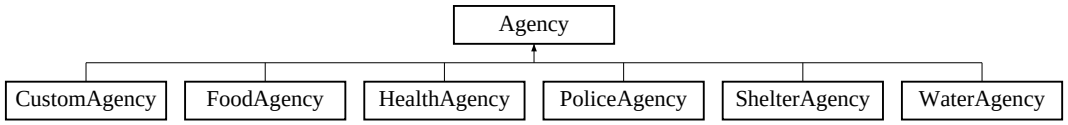
- Activity.h
- Activity.cpp

4.4 Agency Class Reference

Abstract class containing the basic functionality for a Stratmas Agency.

`#include <Agency.h>`

Inheritance diagram for Agency::



Public Member Functions

- **Agency** (const std::vector< **AgencyTeam** * > &teams, **Grid** &g)

  Constructor.

- virtual ~**Agency** ()

  Destructor.

- void **addTeam** (**AgencyTeam** &team)

  Adds a team to this Agency.

- void **removeTeam** (**AgencyTeam** &team)

  Removes a team from this Agency.

- void **aggregateCapacity** ()

  Updates the total capacity and response time for this Agency.

- virtual void **act** (**Time** now)

  Agency over all default behaviour.

Protected Member Functions

- virtual bool **severeProblem** ()=0

  Should return true if there is a severe problem of the type a certain Agency is interesed in.

- int **cluster** (int inNumClusters, std::vector< **LatLng** > &outCenters)

  Finds centers of clusters of the specified type.

- void **orderTeamsToClusters** (int nTeams, int firstTeam)

  Assign teams to clusters.

- virtual void **setTeamsGoals** ()

  Agency default behaviour for positioning teams.

Protected Attributes

- **Time mLastUpdate**

  Simulation(p. 424) *time when last updated.*

- int **mIntervallDays**

  The number of days between rechecking of the need, i.e. applying the cluster algorithm to the grid.

- std::string **mType**

  The type of this agency as a string.

- std::vector< **AgencyTeam** * > **mTeams**

  Vector containing this Agency's teams.

- int **mNumTeams**

  The number of teams in this Agency.

- double **mCapacityPPD**

  The total capacity of this Agency in persons per day.

- int **mResponseDays**

  The response time in days.

- **Grid** & **mGrid**

  A reference to the **Grid**(p. 227).

- int * **mVindex**

  Array used for clustering. Keeps track of which cell that belongs to which cluster.

- double * **mVw**

  Array used for clustering. Keeps track of the weights for different cells.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Agency** &a)

  For debugging purposes.

4.4.1 Detailed Description

Abstract class containing the basic functionality for a Stratmas Agency.

Author:
Per Alexius

Date:
2006/10/02 16:01:25

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Agency::Agency (const std::vector< AgencyTeam * > & *teams*, Grid & *g*)

Constructor.

Parameters:

teams A vector containing this Agency's teams.

g A reference to the **Grid**(p. 227).

4.4.3 Member Function Documentation

4.4.3.1 void Agency::act (Time *now*) [virtual]

Agency over all default behaviour.

The default behaviour is as follows:

For all teams that are not operational - set their start time.

For operational teams - Check if current time is later than their start time and if it is - consider them active teams.

For all active teams - calculate their need, divide the resources evenly among all teams and let each team supply their resources.

Parameters:

now The current simulation time.

Reimplemented in **ShelterAgency** (p. 421), and **CustomAgency** (p. 134).

4.4.3.2 void Agency::addTeam (AgencyTeam & *team*)

Adds a team to this Agency.

Parameters:

team The team to add.

4.4.3.3 int Agency::cluster (int *inNumClusters*, std::vector< LatLng > & *outCenters*) [protected]

Finds centers of clusters of the specified type.

This function requires that mVindex and mVw is set, which is done by the **severeProblem()**(p. 28) function in the respective subclass.

Parameters:

inNumClusters The maximum number of clusters to be sought for.

outCenters An array of positions indicating where the centers of the found clusters are located.

Returns:

The number of clusters found. Not necessarily the same as inNumClusters.

4.4.3.4 void Agency::orderTeamsToClusters (int *nTeams*, int *firstTeam*) [protected]

Assign teams to clusters.

If there are more teams than clusters - assign more than one team to each cluster.

Parameters:

nTeams Numer of teams to assign.

firstTeam Index in the mTeams vector of the first team to assign.

4.4.3.5 void Agency::removeTeam (AgencyTeam & *team*)

Removes a team from this Agency.

Notice that the team itself is not deleted.

Parameters:

team The team to remove.

4.4.3.6 void Agency::setTeamsGoals () [protected, virtual]

Agency default behaviour for positioning teams.

The default behaviour is as follows: Assign one team to each camp. After that, if we have a severe problem - assign excess teams to clusters. If we have more excess teams than clusters - assign more than one team to each cluster. Both food, water, health and shelter agencies follow this behaviour.

Reimplemented in **PoliceAgency** (p. 323), and **CustomAgency** (p. 134).

4.4.3.7 virtual bool Agency::severeProblem () [protected, pure virtual]

Should return true if there is a severe problem of the type a certain Agency is interesed in.

Returns:

True if there is a severe problem of the type this Agency is interested in, false otherwise.

Implemented in **FoodAgency** (p. 218), **WaterAgency** (p. 573), **ShelterAgency** (p. 421), **HealthAgency** (p. 259), **PoliceAgency** (p. 323), and **CustomAgency** (p. 134).

4.4.4 Friends And Related Function Documentation

4.4.4.1 std::ostream& operator<< (std::ostream & *o*, const Agency & *a*) [friend]

For debugging purposes.

Parameters:

o The ostream to write to.

a The Agency to write.

Returns:

The provided ostream with the Agency written to it.

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

- Agency.h
- Agency.cpp

4.5 AgencyFactory Class Reference

Factory for creating Agencies.

`#include <AgencyFactory.h>`

Static Public Member Functions

- void **createAgencies** (**Grid** &grid, const std::vector< **AgencyTeam** * > &teams, std::vector< **Agency** * > &ioAgencies)

  Creates Agencies based on a vector of already created AgencyTeams.

- void **addTeam** (**Grid** &grid, **AgencyTeam** &team, std::vector< **Agency** * > &ioAgencies)

  Adds a team to an **Agency**(p. 25). *If the correct type of* **Agency**(p. 25) *does not yet exist it will be created.*

- void **removeTeam** (**AgencyTeam** &team, std::vector< **Agency** * > &ioAgencies)

  Removes a team from an **Agency**(p. 25).

Private Attributes

- **Grid** & **mGrid**

  Reference(p. 378) *to the* **Grid**(p. 227).

4.5.1 Detailed Description

Factory for creating Agencies.

Author:
Per Alexius

Date:
2006/03/06 09:18:04

4.5.2 Member Function Documentation

4.5.2.1 void AgencyFactory::addTeam (Grid & *grid*, AgencyTeam & *team*, std::vector< Agency * > & *ioAgencies*) [static]

Adds a team to an **Agency**(p. 25). If the correct type of **Agency**(p. 25) does not yet exist it will be created.

Notice that the team may be of any type. The AgencyFactory adds the team to the correct **Agency**(p. 25).

Parameters:
grid A reference to the **Grid**(p. 227).

team The **AgencyTeam**(p. 32) to add.

ioAgencies On return this vector contains the same Agencies as before either with the provided team added to the correct **Agency**(p. 25) or a new **Agency**(p. 25) created with the provided team in it (if the correct type of **Agency**(p. 25) did not exist).

4.5.2.2 void AgencyFactory::createAgencies (Grid & *grid*, const std::vector< AgencyTeam * > & *teams*, std::vector< Agency * > & *ioAgencies*) [static]

Creates Agencies based on a vector of already created AgencyTeams.

Notice that the teams may be of any type. The AgencyFactory sorts out the different types and creates the corresponding Agencies.

Parameters:

grid A reference to the **Grid**(p. 227).

teams A vector containing the teams to create Agencies for.

ioAgencies On return this vector contains the newly created Agencies.

4.5.2.3 void AgencyFactory::removeTeam (AgencyTeam & *team*, std::vector< Agency * > & *ioAgencies*) [static]

Removes a team from an **Agency**(p. 25).

Notice that the team may be of any type. The AgencyFactory removes the team from the correct **Agency**(p. 25).

Parameters:

team The **AgencyTeam**(p. 32) to remove.

ioAgencies On return this vector contains the same Agencies as before but with the provided team removed from the correct **Agency**(p. 25).

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

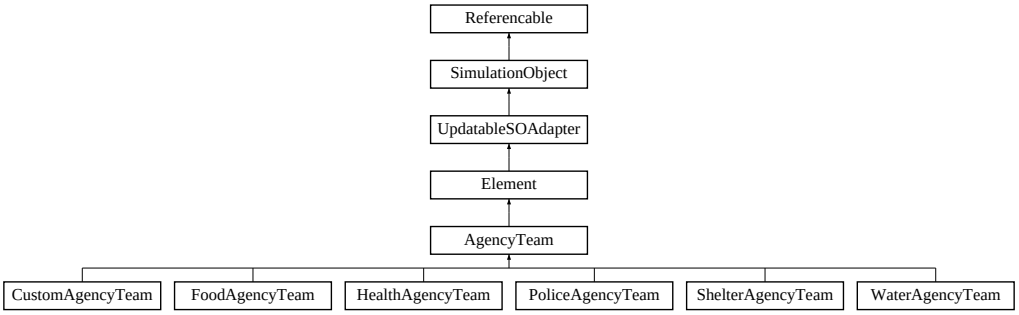
- AgencyFactory.h
- AgencyFactory.cpp

4.6 AgencyTeam Class Reference

Abstract class that is inherited by all AgencyTeams.

`#include <AgencyTeam.h>`

Inheritance diagram for AgencyTeam::



Public Member Functions

- **AgencyTeam** (const **DataObject** &d)
  Constructor that creates an AgencyTeam from the provided **DataObject**(p. 145).
- virtual ~**AgencyTeam** ()
  Destructor.
- void **deploy** ()
  Deploys the team.
- void **depart** ()
  Makes the team depart.
- bool **deployed** () const
  Accessor for the deployed flag.
- bool **departed** () const
  Accessor for the departed flag.
- bool **ownInitiative** () const
  Accessor for the own initiative flag.
- void **setAgency** (**Agency** *agency)
  Sets the **Agency**(p. 25) *this team belons to.*
- void **prepareForSimulation** (**Grid** &grid, const **GridDataHandler** &gdh)
  Prepares this **SimulationObject**(p. 429) *for simulation.*
- virtual void **extract** (**Buffer** &buf) const
  Extracts data from this object to the **Buffer**(p. 67).

- void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- bool **operational** () const

  Returns true if this team is operational, i.e. has been given a start time and a goal.

- bool **present** () const

  Checks if this AgencyTeam is currently present in the simulation, i.e. if it is operational.

- bool **hasStartTime** () const

  Returns true if this team has been given a start time.

- **Time startTime** () const

  Accessor for the start time.

- void **setStartTime** (**Time** t)

  Mutator for the start time.

- void **setGoal** (**LatLng** goal)

  Mutator for the team's goal.

- void **setGoal** (**Camp** *camp)

  Mutator for the team's goal. Sets this teams goal to the camp pointed to by the pointer camp.

- void **setCapacity** (double cap)

  Mutator for the team's capacity.

- double **getCapacity** () const

  Accessor for the team's capacity.

- double **getResponseTime** () const

  Accessor for the team's responseTime.

- **Time getDeployTime** () const
- **Time getDepartTime** () const
- virtual double **calculateNeed** ()

  Calculates the current need of this team. Overridden by teams that have need-based behaviour.

- virtual void **act** (**Time** now)=0

  Performs the actions of this team. Must be implemented by all subclasses.

Protected Member Functions

- bool **canWorkAt** (**Shape** &loc) const

  Checks if the team can go to the given location, e.g. if the violence level is low enough.

Protected Attributes

- **Agency** * **mAgency**

  The **Agency**(p. 25) *this team belongs to.*

- **Camp** * **mCamp**

  If goal is a camp this pointer points at that camp.

- **LatLng mGoal**

  Goal that team moves towards.

- **Time mStartTime**

  Day when team starts operating.

- double **mCapacityPPD**

  Team's capacity in persons per day.

- double **mResponseTimeSecs**

  Response time in seconds.

- **Grid** * **mGrid**

  A reference to the **Grid**(p. 227).

- const **GridDataHandler** * **mGridDataHandler**

  A reference to the **GridDataHandler**(p. 249).

- bool **mHasStartTime**

  True if the team has been given a start time.

- double **mViolenceThreshold**

  The team will not go where violence > mViolenceThreshold.

- bool **mDeployed**

  Flag indicating whether or not this team is deployed.

- bool **mDeparted**

  Flag indicating whether or not this team has departed.

- **Time mDeployTime**

The deploy time.

- **Time mDepartTime**

  The depart time.

- bool **mOwnInitiative**

  Flag indicating the behavior of the team.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **AgencyTeam** &a)

  For debugging purposes.

4.6.1 Detailed Description

Abstract class that is inherited by all AgencyTeams.

Author:
Per Alexius

Date Date
2006/10/10 09:35:59

4.6.2 Constructor & Destructor Documentation

4.6.2.1 AgencyTeam::AgencyTeam (const DataObject & *d*)

Constructor that creates an AgencyTeam from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.6.3 Member Function Documentation

4.6.3.1 virtual void AgencyTeam::act (Time *now*) [pure virtual]

Performs the actions of this team. Must be implemented by all subclasses.

Parameters:
now The current simulation time.

Implemented in **FoodAgencyTeam** (p. 220), **WaterAgencyTeam** (p. 575), **ShelterAgencyTeam** (p. 423), **HealthAgencyTeam** (p. 261), **PoliceAgencyTeam** (p. 325), and **CustomAgencyTeam** (p. 136).

4.6.3.2 void AgencyTeam::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:

toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 564).

Reimplemented in **CustomAgencyTeam** (p. 137).

4.6.3.3 virtual double AgencyTeam::calculateNeed () [inline, virtual]

Calculates the current need of this team. Overridden by teams that have need-based behaviour.

Returns:

The current need of this team.

Reimplemented in **FoodAgencyTeam** (p. 220), **WaterAgencyTeam** (p. 575), and **HealthAgencyTeam** (p. 261).

4.6.3.4 bool AgencyTeam::canWorkAt (Shape & *loc*) const [protected]

Checks if the team can go to the given location, e.g. if the violence level is low enough.

Parameters:

goal The goal.

Returns:

True if the team can go, false otherwise.

4.6.3.5 bool AgencyTeam::departed () const [inline]

Accessor for the departed flag.

Returns:

The state of the departed flag..

4.6.3.6 bool AgencyTeam::deployed () const [inline]

Accessor for the deployed flag.

Returns:

The state of the deployed flag..

4.6.3.7 void AgencyTeam::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Reimplemented from **Element** (p. 182).

Reimplemented in **CustomAgencyTeam** (p. 137).

4.6.3.8 double AgencyTeam::getCapacity () const [inline]

Accessor for the team's capacity.

Returns:
cap The team's capacity

4.6.3.9 double AgencyTeam::getResponseTime () const [inline]

Accessor for the team's responseTime.

Returns:
cap The team's responseTime

4.6.3.10 bool AgencyTeam::hasStartTime () const [inline]

Returns true if this team has been given a start time.

Returns:
True if this team has been given a start time.

4.6.3.11 void AgencyTeam::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **Element** (p. 182).

Reimplemented in **CustomAgencyTeam** (p. 137).

4.6.3.12 bool AgencyTeam::operational () const [inline]

Returns true if this team is operational, i.e. has been given a start time and a goal.

Returns:
True if this team is operational, i.e. has been given a start time and a goal, false otherwise.

4.6.3.13 bool AgencyTeam::ownInitiative () const [inline]

Accessor for the own initiative flag.

Returns:
The state of the own initiative flag..

4.6.3.14 void AgencyTeam::prepareForSimulation (Grid & *grid*, const GridDataHandler & *gdh*) [inline]

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Parameters:
grid The **Grid**(p. 227).

4.6.3.15 bool AgencyTeam::present () const [inline, virtual]

Checks if this AgencyTeam is currently present in the simulation, i.e. if it is operational.

Returns:
True if this AgencyTeam is present, false otherwise.

Implements **Element** (p. 182).

4.6.3.16 void AgencyTeam::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:
toRemove The **Reference**(p. 378) to the object to remove.
initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

Reimplemented in **CustomAgencyTeam** (p. 137).

4.6.3.17 void AgencyTeam::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Element** (p. 183).

Reimplemented in **CustomAgencyTeam** (p. 138).

4.6.3.18 void AgencyTeam::setAgency (Agency * *agency*) [inline]

Sets the **Agency**(p. 25) this team belons to.

Parameters:
agency The **Agency**(p. 25) this team shoul belong to.

4.6.3.19 void AgencyTeam::setCapacity (double *cap*) [inline]

Mutator for the team's capacity.

Parameters:
cap The team's capacity

4.6.3.20 void AgencyTeam::setGoal (Camp * *camp*)

Mutator for the team's goal. Sets this teams goal to the camp pointed to by the pointer camp.

Parameters:
camp The **Camp**(p. 74).

4.6.3.21 void AgencyTeam::setGoal (LatLng *goal*)

Mutator for the team's goal.

Parameters:
goal The team's goal.

4.6.3.22 void AgencyTeam::setStartTime (Time *t*) [inline]

Mutator for the start time.

Parameters:
t The start time.

4.6.3.23 Time AgencyTeam::startTime () const [inline]

Accessor for the start time.

Returns:
The start time.

4.6.4 Friends And Related Function Documentation

4.6.4.1 std::ostream& operator<< (std::ostream & *o*, const AgencyTeam & *a*) [friend]

For debugging purposes.

Parameters:

o The ostream to write to.

a The AgencyTeam to write.

Returns:

The provided ostream with the AgencyTeam written to it.

4.6.5 Member Data Documentation

4.6.5.1 bool AgencyTeam::mOwnInitiative [protected]

Flag indicating the behavior of the team.

A 'true' value means that the team behaves as in old Stratmas, i.e jumps around to the places where the clustering algorithm detects clusters of 'problem cells'. A 'false' value means that the team will stand still on the initial location given to it.

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

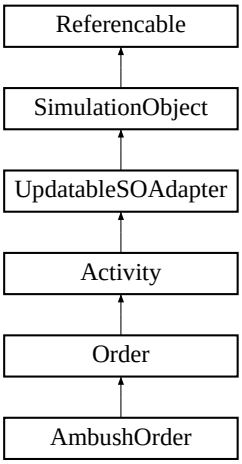
- AgencyTeam.h
- AgencyTeam.cpp

4.7 AmbushOrder Class Reference

The AmbushOrder.

`#include <Activity.h>`

Inheritance diagram for AmbushOrder::



Public Types

- enum **eAmbushState** { **eHide**, **eAmbush** }

  Enumeration for the state of an AmbushOrder.

Public Member Functions

- **AmbushOrder** (const **DataObject** &d)

  Creates an AmbushOrder object from the provided **DataObject**(p. 145).

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- bool **isActive** (**Time** t)

  Checks if this activity is active at time t.

- bool **oneAmbush** () const

  Checks if the startAmbush and endAmbush times are equal.

- void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

- double **combatFactor** () const

  Accessor for the combat factor.

- int **state** () const

  Accessor for the state.

Protected Attributes

- int **mState**

  The state of this AmbushOrder.

- bool **mTimeForAmbush**

  True when time is in ambush intervall or when time is after mStartTime and **oneAmbush()**(p. 43) *== true and mOneAmbushPerformed == false.*

- bool **mOneAmbushPerformed**

  If this is a 'oneAmbush' order e.g. if mStartAmbush and mEndAmbush are equal, then this flag is set to true when one ambush has been performed.

- **Time mEnd**

  The end time of the order (the 'hide' state).

- **Time mStartAmbush**

  The start time for the ambush activity.

- **Time mEndAmbush**

  The end time for the ambush activity.

4.7.1 Detailed Description

The AmbushOrder.

Author:
Per Alexius

Date Date
2006/07/19 07:04:26

4.7.2 Constructor & Destructor Documentation

4.7.2.1 AmbushOrder::AmbushOrder (const DataObject & *d*)

Creates an AmbushOrder object from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) from which to create this object.

4.7.3 Member Function Documentation

4.7.3.1 double AmbushOrder::combatFactor () const [inline, virtual]

Accessor for the combat factor.

Returns:
The combat factor.

Implements **Order** (p. 310).

4.7.3.2 void AmbushOrder::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Reimplemented from **Order** (p. 310).

4.7.3.3 bool AmbushOrder::isActive (Time *t*) [virtual]

Checks if this activity is active at time t.

Parameters:
t The time for which to check.

Returns:
True if this activity is active at the specified time.

Reimplemented from **Order** (p. 310).

4.7.3.4 void AmbushOrder::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **Order** (p. 311).

4.7.3.5 bool AmbushOrder::oneAmbush () const [inline]

Checks if the startAmbush and endAmbush times are equal.

Returns:
True if the startAmbush and endAmbush times are equal.

4.7.3.6 void AmbushOrder::perform (Element * *e*, double *fraction* = 1.0) [virtual]

Performs this **Activity**(p. 21).

Parameters:

e The **Element**(p. 180) that should perform this **Activity**(p. 21).

fraction The fraction of the performers total capacity that this activity is performed with.

Implements **Activity** (p. 23).

4.7.3.7 void AmbushOrder::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Order** (p. 312).

4.7.3.8 int AmbushOrder::state () const [inline]

Accessor for the state.

Returns:

The state.

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

- Activity.h
- Activity.cpp

4.8 AmbushRecord Class Reference

Record for holding information of ambushs.

`#include <Unit.h>`

Public Member Functions

- **AmbushRecord** ()
  Default constructor.
- **AmbushRecord** (const **AmbushRecord** &ar)
  Copy constructor.
- bool **active** () const
  Checks if this record is active, i.e. if there are any potential victims.
- void **addUnit** (**Unit** *u)
  Adds a potential victim.
- const std::set< **Unit** * > & **units** () const
  Accessor for the vector containing the potential victims.
- double **damage** () const
  Accessor for the damage.
- void **addDamage** (double damage)
  Adds some damage to this record.
- void **selectTarget** ()
  Selects a target unit among the registered potential target units.

Private Attributes

- std::set< **Unit** * > **mUnits**
  The set of units that may be affected by an ambush in the cell this AmbushRecord refers to.
- double **mDamage**
  The damage this unit will cause in this cell if an ambush occurs.

4.8.1 Detailed Description

Record for holding information of ambushs.

A unit has an AmbushRecord for each cell it should try to perform an ambush in.

Author:
Per Alexius

Date **Date**
2006/04/21 15:54:52

4.8.2 Constructor & Destructor Documentation

4.8.2.1 AmbushRecord::AmbushRecord (const AmbushRecord & *ar*) [inline]

Copy constructor.

Parameters:
ar The record to copy.

4.8.3 Member Function Documentation

4.8.3.1 bool AmbushRecord::active () const [inline]

Checks if this record is active, i.e. if there are any potential victims.

Returns:
True if the record is active, false otherwise.

4.8.3.2 void AmbushRecord::addDamage (double *damage*) [inline]

Adds some damage to this record.

Parameters:
damage The damage to add.

4.8.3.3 void AmbushRecord::addUnit (Unit * *u*) [inline]

Adds a potential victim.

Parameters:
u The **Unit**(p. 546) to add.

4.8.3.4 double AmbushRecord::damage () const [inline]

Accessor for the damage.

Returns:
The damage.

4.8.3.5 void AmbushRecord::selectTarget ()

Selects a target unit among the registered potential target units.

The probability for a unit to be chosen as target is proportional to the number of personnel for that unit. All units but the selected unit are removed from the record.

4.8.3.6 const std::set<Unit*>& AmbushRecord::units () const [inline]

Accessor for the vector containing the potential victims.

Returns:
The vector containing the potential victims.

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

- Unit.h
- Unit.cpp

4.9 AreaHandler Class Reference

Class that finds out which cells a certain shape covers on a certain grid.

`#include <AreaHandler.h>`

Public Member Functions

- **AreaHandler** (int rows, int cols, double t, double b, double l, double r, double dx, double dy, double cs)

  Creates an areahandler for a grid with the specified parameters.

- **GridPos cell** (const **ProjCoord** p) const

  Gets the position in the grid for the cell that contains the specified point.

- void **cells** (const **Polygon** &inP, std::list< **GridPos** > &outCells) const

  Returns a list containing the grid positions of all cells covered by the provided **Polygon**(p. 326).

- void **cells** (const **Circle** &inC, std::list< **GridPos** > &outCells) const

  Returns a list containing the grid positions of all cells covered by the provided **Circle**(p. 86).

Private Member Functions

- void **BuildGET** (std::list< **GridPos** > &VertexList, **EdgeState** *NextFreeEdgeStruc, int XOffset, int YOffset) const

  Creates a Global Edge Table (GET) in the buffer pointed to by NextFreeEdgeStruc from the vertex list.

- void **MoveXSortedToAET** (int YToMove) const

  Moves all edges that start at the specified Y coordinate from the GET to the AET, maintaining the X sorting of the AET.

- void **ScanOutAET** (int YToScan, std::list< **GridPos** > &outInterior) const

  Fills the scan line described by the current AET at the specified Y coordinate.

- void **AdvanceAET** () const

  Advances each edge in the AET by one scan line. Removes edges that have been fully scanned.

- void **XSortAET** () const

  Sorts all edges currently in the active edge table into ascending order of current X coordinates.

- **GridPos cellPos** (double x, double y) const

  Gets the position in the grid for the cell that contains the specified point.

- **GridPos cellPos** (const gpc_vertex &p) const

  Gets the position in the grid for the cell that contains the specified point.

- double **borderBetweenRows** (int r1, int r2) const

  Gets the y-coordinate of the line separating the two provided rows.

- void **addCellsInCurrentRow** (std::list< **GridPos** > &l, int dc) const

  Adds specified number of cells in the current row.

- void **addCellsInCurrentCol** (std::list< **GridPos** > &l, int dr) const

  Adds specified number of cells in the current column.

- void **polygonBoundaryToCellBoundary** (gpc_vertex *inP, int inNumPoints, std::list< **GridPos** > &outB) const

  Gets the cells that a polygon boundary overlaps (not the interior).

- void **splitBoundary** (const std::list< **GridPos** > &inB, std::list< std::list< **GridPos** > > &outB) const

  Splits the provided boundary so that all loops are stored as separate boundaries.

- int **getInterior** (std::list< **GridPos** > &VertexList, std::list< **GridPos** > &outInterior) const

  Gets the interior grid positions of the provided exterior.

Private Attributes

- int **mRows**

  Number of rows in the grid.

- int **mCols**

  Number of columns in the grid.

- double **mTop**

  Top coordinate.

- double **mBottom**

  Bottom coordinate.

- double **mLeft**

  Leftmost coordinate.

- double **mRight**

  Rightmost coordinate.

- double **mDx**

  The width of a cell.

- double **mDy**

  The height of a cell.

- double **mCellSideMeters**

  The cells side in meters.

4.9.1 Detailed Description

Class that finds out which cells a certain shape covers on a certain grid.

The algorithm for finding the interior of a polygon is taken from Michel Abrash's Graphics Programming Black Book Special Edition (ISBN 1-57610-174-6).

Author:
Per Alexius

Date:
2005/06/15 09:28:18

4.9.2 Constructor & Destructor Documentation

4.9.2.1 AreaHandler::AreaHandler (int *rows*, int *cols*, double *t*, double *b*, double *l*, double *r*, double *dx*, double *dy*, double *cs*) [inline]

Creates an areahandler for a grid with the specified parameters.

Parameters:
rows Number of rows in the grid.
cols Number of columns in the grid.
t Top coordinate.
b Bottom coordinate.
l Leftmost coordinate.
r Rightmost coordinate.
dx The width of a cell.
dy The height of a cell.
cs The cells side in meters.

4.9.3 Member Function Documentation

4.9.3.1 void AreaHandler::addCellsInCurrentCol (std::list< GridPos > & *l*, int *dr*) const [inline, private]

Adds specified number of cells in the current column.

The current column is the column of the last element in the list.

Parameters:
l The list which last element indicates the current column and to which to add cells.
dr The number of cells to add - positive for down, negative for up.

4.9.3.2 void AreaHandler::addCellsInCurrentRow (std::list< GridPos > & *l*, int *dc*) const [inline, private]

Adds specified number of cells in the current row.

The current row is the row of the last element in the list.

Parameters:

l The list which last element indicates the current row and to which to add cells.

dc The number of cells to add - positive for right, negative for left.

4.9.3.3 double AreaHandler::borderBetweenRows (int *r1*, int *r2*) const [inline, private]

Gets the y-coordinate of the line separating the two provided rows.

Parameters:

r1 The first row.

r2 The second row.

Returns:

The y-coordinate of the line separating the two provided rows.

4.9.3.4 void AreaHandler::BuildGET (std::list< GridPos > & *VertexList*, EdgeState * *NextFreeEdgeStruc*, int *XOffset*, int *YOffset*) const [private]

Creates a Global Edge Table (GET) in the buffer pointed to by NextFreeEdgeStruc from the vertex list.

Edge endpoints are flipped, if necessary, to guarantee all edges go top to bottom. The GET is sorted primarily by ascending Y start coordinate, and secondarily by ascending X start coordinate within edges with common Y coordinates

Parameters:

VertexList A list of vertices.

NextFreeEdgeStruc Allocated memory for all **EdgeState**(p. 178) structs.

XOffset Horizontal offset.

YOffset Verical offset.

4.9.3.5 GridPos AreaHandler::cell (const ProjCoord *p*) const

Gets the position in the grid for the cell that contains the specified point.

Parameters:

p The point.

Returns:

The position in the grid for the cell that contains the specified point.

4.9.3.6 GridPos AreaHandler::cellPos (const gpc_vertex & *p*) const [inline, private]

Gets the position in the grid for the cell that contains the specified point.

Parameters:

p The point.

Returns:

The position in the grid for the cell that contains the specified point.

4.9.3.7 GridPos AreaHandler::cellPos (double *x*, double *y*) const [inline, private]

Gets the position in the grid for the cell that contains the specified point.

Parameters:

x The x coordinate of the point.

y The y coordinate of the point.

Returns:

The position in the grid for the cell that contains the specified point.

4.9.3.8 void AreaHandler::cells (const Circle & *inC*, std::list< GridPos > & *outCells*) const

Returns a list containing the grid positions of all cells covered by the provided **Circle**(p. 86).

Parameters:

inC The **Circle**(p. 86) to get the grid positions for.

outCells A list that on return contains pointers to all cells covered by the provided **Circle**(p. 86).

4.9.3.9 void AreaHandler::cells (const Polygon & *inP*, std::list< GridPos > & *outCells*) const

Returns a list containing the grid positions of all cells covered by the provided **Polygon**(p. 326).

Parameters:

inP The **Polygon**(p. 326) to get the grid positions for.

outCells A list that on return contains the grid positions of all cells covered by the provided **Polygon**(p. 326).

4.9.3.10 int AreaHandler::getInterior (std::list< GridPos > & *VertexList*, std::list< GridPos > & *outInterior*) const [private]

Gets the interior grid positions of the provided exterior.

Parameters:

VertexList A list of grid positions constituting the boundary of the polygon to get the interior for.

outInterior A list of grid positions that on return contains the same values as on entry plus the interior grid positions.

4.9.3.11 void AreaHandler::MoveXSortedToAET (int *YToMove*) const [private]

Moves all edges that start at the specified Y coordinate from the GET to the AET, maintaining the X sorting of the AET.

Parameters:

YToMove The y-coordinate for which to move edges.

4.9.3.12 void AreaHandler::polygonBoundaryToCellBoundary (gpc_vertex * *inP*, int *inNumPoints*, std::list< GridPos > & *outB*) const [private]

Gets the cells that a polygon boundary overlaps (not the interior).

Does: For each edge in the polygon - check which cells it crosses and store those cells in a linked list. At exit the linked list contains all cells in the boundary in order such that there is an edge between outB[i] and outB[i+1].

Idea: For each line (p1, p2) between the points p1 and p2 do:

Find the cells c1 and c2 that contains the points p1 and p2.

If c1 == c2

Head on to the next point

Else

Calculate number of rows (dr) and cols (dc) the line passes.

If dr or dc == 0

We have a trivial case so add edges between cells in the

current row or column between c1 and c2

Else

For each row that (p1, p2) passes find out in which column c

the intersection is and add edges between all cells on that

row between the current column oldc and c. Also add an edge

between the cells on current row and the next row, in column c.

End

End

Parameters:

inP The vertices of the polygon.

inNumPoints The number of vertices in the polygon.

outB A list that on return contains the positions of the cells that the provided polygon boundary overlaps.

4.9.3.13 void AreaHandler::ScanOutAET (int *YToScan*, std::list< GridPos > & *outInterior*) const [private]

Fills the scan line described by the current AET at the specified Y coordinate.

Parameters:

YToScan The y-coordinate for which to fill the scan line.

outInterior A list that on return contains the same values as on entry plus the cells on the line just scanned.

4.9.3.14 void AreaHandler::splitBoundary (const std::list< GridPos > & *inB*, std::list< std::list< GridPos > > & *outB*) const [private]

Splits the provided boundary so that all loops are stored as separate boundaries.

Does: Splits inB so that all loops are stored as a separate list of cells

in outB. If inB contains no loops the first element in outB will contain

a list identical to inB

Idea: For each cell c in l (l is a copy of inB)

If c was visited before e.g if we have encountered a loop

Store cells that are part of the loop in a new list

Remove all cells that are part of the loop except c from l

Unmark all cells that are part of the loop except c

Parameters:

inB A list of grid positions constituting a contour of a polygon.

outB A list of lists that on return contains one list of grid positions for each loop found.

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

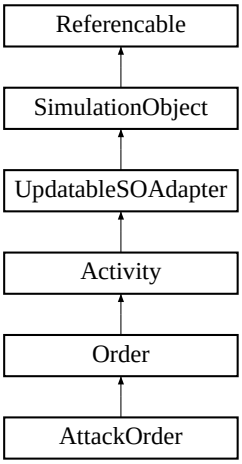
- AreaHandler.h
- AreaHandler.cpp

4.10 AttackOrder Class Reference

The AttackOrder.

`#include <Activity.h>`

Inheritance diagram for AttackOrder::



Public Member Functions

- **AttackOrder** ()

  Default constructor.

- **AttackOrder** (const **DataObject** &d)

  Creates an AttackOrder from the provided **DataObject**(p. 145).

- void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

- double **combatFactor** () const

  Accessor for the combat factor.

4.10.1 Detailed Description

The AttackOrder.

Author:
Per Alexius

Date **Date**
2006/07/19 07:04:26

4.10.2 Constructor & Destructor Documentation

4.10.2.1 AttackOrder::AttackOrder (const DataObject & *d*) [inline]

Creates an AttackOrder from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this order from.

4.10.3 Member Function Documentation

4.10.3.1 double AttackOrder::combatFactor () const [inline, virtual]

Accessor for the combat factor.

Returns:

The combat factor.

Implements **Order** (p. 310).

4.10.3.2 void AttackOrder::perform (Element * *e*, double *fraction* = 1.0) [virtual]

Performs this **Activity**(p. 21).

Parameters:

e The **Element**(p. 180) that should perform this **Activity**(p. 21).

fraction The fraction of the performers total capacity that this activity is performed with.

Implements **Activity** (p. 23).

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

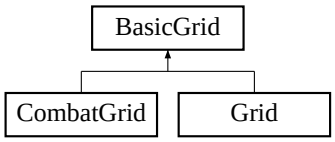
- Activity.h
- Activity.cpp

4.11 BasicGrid Class Reference

This class represents basic characteristics for a grid overlayed by a map.

`#include <BasicGrid.h>`

Inheritance diagram for BasicGrid::



Public Member Functions

- **BasicGrid** (const **Map** &amap, double cellSizeMeters)
  Creates a BasicGrid for the provided **Map**(p. 292) *with the specified cell size.*
- virtual ~**BasicGrid** ()
  Destructor.
- int **rows** () const
  Accessor for the number of rows.
- int **cols** () const
  Accessor for the number of columns.
- int **cells** () const
  Accessor for the number of cells.
- int **active** () const
  Accessor for the number of active cells.
- bool **isActive** (int r, int c) const
  Checks if the specified position is inside active grid.
- int **posToActive** (int r, int c) const
  Gets the index in the active array from a position in the grid.
- int **activeToIndex** (int activeIndex) const
  *Gets the position (r * mCols + c) from an index in the active array.*
- double **cellSideMeters** () const
  Accessor for the cell side.
- double **cellAreaKm2** () const
  Accessor for the cell area.
- const **Map** & **map** () const

Accessor for the map.

- const double * **cellPosLatLng** () const
  Accessor for the cell positions array.
- const double * **cellPosProj** () const
  Accessor for the cell positions array.
- **GridPos cell** (const **ProjCoord** p) const
  Gets the position in the grid for the cell that contains the specified point.
- **GridPos cell** (const **LatLng** p) const
  Gets the position in the grid for the cell that contains the specified point.
- void **cells** (const **Polygon** &inP, std::list< **GridPos** > &outCells) const
  Returns a list containing the grid positions of all cells covered by the provided **Polygon**(p. 326).
- void **cells** (const **Circle** &inC, std::list< **GridPos** > &outCells) const
  Returns a list containing the grid positions of all cells covered by the provided **Circle**(p. 86).
- **LatLng center** (int r, int c) const
  Gets the center coordinate (lat, lag) for the specified cell.

Protected Attributes

- int **mRows**
  Number of rows.
- int **mCols**
  Number of columns.
- int **mCells**
  Number of cells.
- int **mActive**
  Number of active cells.
- double **mCellSideMeters**
  Cell side in meters.
- double **mCellArea**
  Area of each cell.
- double **mTop**
  Top coordinate.
- double **mBottom**
  Bottom coordinate.

- double **mLeft**

  Leftmost coordinate.

- double **mRight**

  Rightmost coordinate.

- double **mDx**

  The width of a cell.

- double **mDy**

  The height of a cell.

- **AreaHandler** * **mAH**

  The **AreaHandler**(p. 48) *for this* **Grid**(p. 227).

- const **Map** & **mMap**

  The **Map**(p. 292) *used when creating this* **Grid**(p. 227).

- double * **mCellPosLatLng**

  An array for storing cell positions in.

- double * **mCellPosProj**

  An array for storing cell positions in.

Private Member Functions

- void **sizeGrid** (const **Map** &m)

  Sets the grid size parameters based on the provided **Map**(p. 292).

- void **setCellPositions** ()

  Initializes the array that holds the coordinates of the cornerpoints of all cells.

Private Attributes

- int * **mIndexToActive**

  *Maps a cell's position in the grid (r * mCols + c) to its position in the active array. Inactive cells are marked with kInactive. The size of this array is thus mCells.*

- int * **mActiveToIndex**

  *Maps a cell's position in the active array to its position in the grid (r * mCols + c). The size of this array is thus mActive.*

Static Private Attributes

- const int **kInactive** = -1

  Inactive cells should be marked with kInactive in the mIndexToActive array.

- const int **kUndefinedActive** = -2

  All cells in the mIndexToActive array are marked as kUndefinedActive to start with. The reason for this is that we must be able to determine which cells the map covers and since it is the map that determines which cells that are active all cells should be active (i.e. kUndefinedActive) when calling **cells()**(p. 62) *for the map. Cells not covered by the map may then be considered inactive.*

Friends

- std::ostream & **operator**<< (std::ostream &o, const **BasicGrid** &g)

  For debugging purposes.

4.11.1 Detailed Description

This class represents basic characteristics for a grid overlayed by a map.

Author:
Per Alexius

Date:
2006/07/19 07:04:27

4.11.2 Constructor & Destructor Documentation

4.11.2.1 BasicGrid::BasicGrid (const Map & *amap*, double *cellSizeMeters*)

Creates a BasicGrid for the provided **Map**(p. 292) with the specified cell size.

Parameters:
amap The map to create the **Grid**(p. 227) for.

cellSizeMeters The side of the cells in meters.

4.11.3 Member Function Documentation

4.11.3.1 int BasicGrid::active () const [inline]

Accessor for the number of active cells.

Returns:
The number of active cells.

4.11.3.2 int BasicGrid::activeToIndex (int *activeIndex*) const [inline]

Gets the position (r * mCols + c) from an index in the active array.

Returns:
The position.

4.11.3.3 GridPos BasicGrid::cell (const LatLng *p*) const

Gets the position in the grid for the cell that contains the specified point.

Parameters:
p The point.

Returns:
The position in the grid for the cell that contains the specified point.

4.11.3.4 GridPos BasicGrid::cell (const ProjCoord *p*) const

Gets the position in the grid for the cell that contains the specified point.

Parameters:
p The point.

Returns:
The position in the grid for the cell that contains the specified point.

4.11.3.5 double BasicGrid::cellAreaKm2 () const [inline]

Accessor for the cell area.

Returns:
The cell area in square meters.

4.11.3.6 const double* BasicGrid::cellPosLatLng () const [inline]

Accessor for the cell positions array.

Returns:
The cell positions array.

4.11.3.7 const double* BasicGrid::cellPosProj () const [inline]

Accessor for the cell positions array.

Returns:
The cell positions array.

4.11.3.8 void BasicGrid::cells (const Circle & *inC*, std::list< GridPos > & *outCells*) const

Returns a list containing the grid positions of all cells covered by the provided **Circle**(p. 86).

Parameters:

inC The **Circle**(p. 86) to get the grid positions for.

outCells A list that on return contains pointers to all cells covered by the provided **Circle**(p. 86).

4.11.3.9 void BasicGrid::cells (const Polygon & *inP*, std::list< GridPos > & *outCells*) const

Returns a list containing the grid positions of all cells covered by the provided **Polygon**(p. 326).

Parameters:

inP The **Polygon**(p. 326) to get the grid positions for.

outCells A list that on return contains the grid positions of all cells covered by the provided **Polygon**(p. 326).

4.11.3.10 int BasicGrid::cells () const [inline]

Accessor for the number of cells.

Returns:

The number of cells.

4.11.3.11 double BasicGrid::cellSideMeters () const [inline]

Accessor for the cell side.

Returns:

The cell side in meters.

4.11.3.12 LatLng BasicGrid::center (int *r*, int *c*) const

Gets the center coordinate (lat, lag) for the specified cell.

Parameters:

r The row of the cell.

c The column of the cell.

Returns:

The coordinate of the center of the specified cell.

4.11.3.13 int BasicGrid::cols () const [inline]

Accessor for the number of columns.

Returns:
The number of columns.

4.11.3.14 bool BasicGrid::isActive (int *r*, int *c*) const [inline]

Checks if the specified position is inside active grid.

Parameters:
r The row.
c The column.

Returns:
True if the specified position is inside active grid, false otherwise.

4.11.3.15 const Map& BasicGrid::map () const [inline]

Accessor for the map.

Returns:
The map.

4.11.3.16 int BasicGrid::posToActive (int *r*, int *c*) const [inline]

Gets the index in the active array from a position in the grid.

Parameters:
r The row.
c The column.

Returns:
The index in the active array.

4.11.3.17 int BasicGrid::rows () const [inline]

Accessor for the number of rows.

Returns:
The number of rows.

4.11.3.18 void BasicGrid::sizeGrid (const Map & *m*) [private]

Sets the grid size parameters based on the provided **Map**(p. 292).

Parameters:
m The **Map**(p. 292).

4.11.4 Friends And Related Function Documentation

4.11.4.1 std::ostream& operator<< (std::ostream & *o*, const BasicGrid & *g*) [friend]

For debugging purposes.

Parameters:

o The stream to write to.

g The BasicGrid to print.

4.11.5 Member Data Documentation

4.11.5.1 double* BasicGrid::mCellPosLatLng [protected]

An array for storing cell positions in.

The array contains lat and lng values for cell 'intersection points' ordered row-wise from top left to bottom right i.e. [lat0 lng0 lat1 lng1...].

4.11.5.2 double* BasicGrid::mCellPosProj [protected]

An array for storing cell positions in.

The array contains projected x and y values for cell 'intersection points' ordered row-wise from top left to bottom right i.e. [x0 y0 x1 y1...].

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

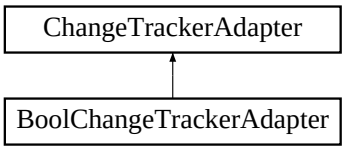
- BasicGrid.h
- BasicGrid.cpp

4.12 BoolChangeTrackerAdapter Class Reference

The BoolChangeTrackerAdapter keeps track of changes in **StratmasBool**(p. 460) objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for BoolChangeTrackerAdapter::



Public Member Functions

- **BoolChangeTrackerAdapter** (**StratmasBool** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 66) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **StratmasBool** & **mObject**

  The adapted **DataObject**(p. 145).

- bool **mLast**

  The last value written.

4.12.1 Detailed Description

The BoolChangeTrackerAdapter keeps track of changes in **StratmasBool**(p. 460) objects.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.12.2 Constructor & Destructor Documentation

4.12.2.1 BoolChangeTrackerAdapter::BoolChangeTrackerAdapter (StratmasBool & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.12.3 Member Function Documentation

4.12.3.1 bool BoolChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 66) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 66) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.12.3.2 ostream & BoolChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

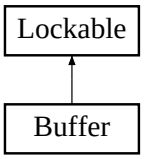
- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.13 Buffer Class Reference

This class is used to store data that should be transfered between the simulation and the clients.

`#include <Buffer.h>`

Inheritance diagram for Buffer::



Public Member Functions

- **Buffer** ()

  Constructor.

- ~**Buffer** ()

  Destructor.

- bool **hasData** () const

  Checks if there is data stored in the buffer.

- const **Grid** & **grid** () const

  Accessor for the **Grid**(p. 227).

- const **CombatGrid** & **combatGrid** () const

  Accessor for the **CombatGrid**(p. 101).

- const **GridDataHandler** & **gridDataHandler** () const

  Accessor for the GridDatahandler.

- int **resetCount** () const

  Accessor for the reset count.

- void **layer** (const std::string &lay, const **Reference** &fac, int size, int32_t *index, double *&outData)

  Fetches process variable values for the specified cells, process variable and faction.

- const **DataObject** * **simulation** () const

  Accessor for the simulation **DataObject**(p. 145).

- const **DataObject** * **originalSimulation** () const

  Accessor for the original simulation **DataObject**(p. 145)*, i.e. the one that the server got initialized with.*

- std::string **simulationName** () const

  Returns the name of the currently initialized simulation.

- void **extractGridData** (**GridDataHandler** *gdh)

  Copies data from the simulation **Grid**(p. 227) *to the Buffer so that it will be accessible for clients.*

- void **transferUpdatesToSimulation** ()

  Transfers the updates from the Buffer to the simulation.

- **Time currentTime** () const

  Accessor for the current simulation time.

- void **currentTime** (const **Time** t)

  Mutator for the current simulation time.

- bool **engineIdle** () const

  Checks if the engin is idle.

- void **engineIdle** (bool b)

  Mutator for the mEngineIdle flag.

- **Time simTime** () const

  Accessor for the time for which the data in the Buffer is valid.

- void **simTime** (**Time** simTime)

  Mutator for the time for which the data in the Buffer is valid.

- void **put** (**DataObject** *d)

  Stores data about a simulation in the Buffer.

- void **put** (std::vector< **Update** * > updates)

  Stores data about updates in the Buffer.

- void **reset** ()

  Resets the buffer to the same state as directly after an initialization.

- **DataObject** * **map** (const **Reference** &ref) const

  Maps the provided **Reference**(p. 378) *to its corresponding* **DataObject**(p. 145).

Private Attributes

- **GridDataHandler** * **mGridDataHandler**

  Handles grid data.

- **DataObject** * **mSimulation**

  The simulation **DataObject**(p. 145).

- **DataObject** * **mSimClone**

  The original simulation **DataObject**(p. 145).

- **Time mSimTime**

Data in the buffer is valid for this time.

- **Time mCurrentTime**
  Current **Engine**(p. 189) *timestep (data not necessarily copied to Buffer).*
- bool **mEngineIdle**
  True if the **Engine**(p. 189) *isn't working on a timestep.*
- int **mResetCount**
  Counts the number of resets.
- std::vector< **Update** * > **mUpdates**
  A vector containing all updates.
- std::string **mSimulationName**
  The name of the currently initialized simulation.

4.13.1 Detailed Description

This class is used to store data that should be transfered between the simulation and the clients.

Author:
Per Alexius

Date **Date**
2006/10/10 09:36:51

4.13.2 Member Function Documentation

4.13.2.1 const CombatGrid & Buffer::combatGrid () const

Accessor for the **CombatGrid**(p. 101).

Returns:
A **Reference**(p. 378) to the **CombatGrid**(p. 101).

4.13.2.2 void Buffer::currentTime (const Time *t*) [inline]

Mutator for the current simulation time.

Parameters:
t The current simulation time.

4.13.2.3 Time Buffer::currentTime () const [inline]

Accessor for the current simulation time.

Returns:
The current simulation time.

4.13.2.4 void Buffer::engineIdle (bool *b*) [inline]

Mutator for the mEngineIdle flag.

Parameters:
b The new state of the mEngineIdle flag.

4.13.2.5 bool Buffer::engineIdle () const [inline]

Checks if the engin is idle.

Returns:
True if the **Engine**(p. 189) is idle, false otherwise.

4.13.2.6 void Buffer::extractGridData (GridDataHandler * *gdh*)

Copies data from the simulation **Grid**(p. 227) to the Buffer so that it will be accessible for clients.

Called by the **Engine**(p. 189) when simulation data should be transfered from the simulation to the Buffer, for example after each timestep and after initialization.

Notice that the call to this function means that the Buffer takes over the responsibility to deallocate the **GridDataHandler**(p. 249).

Parameters:
gdh The **GridDataHandler**(p. 249) created by the **Scenario**(p. 396).

4.13.2.7 const Grid & Buffer::grid () const

Accessor for the **Grid**(p. 227).

Returns:
A **Reference**(p. 378) to the **Grid**(p. 227).

4.13.2.8 const GridDataHandler& Buffer::gridDataHandler () const [inline]

Accessor for the GridDatahandler.

Returns:
The GridDatahandler.

4.13.2.9 bool Buffer::hasData () const [inline]

Checks if there is data stored in the buffer.

Returns:
True if there is data stored in the Buffer.

4.13.2.10 void Buffer::layer (const std::string & *lay*, const Reference & *fac*, int *size*, int32_t * *index*, double *& *outData*)

Fetches process variable values for the specified cells, process variable and faction.

Parameters:

lay The name of the process variable.

fac A **Reference**(p. 378) to the faction.

size The number of cells to fetch values for.

index An array of size elements containing the indices in the active cells array of the cells for which to fetch the values.

outData An array of size elements that on return will contain the values for the specified cells.

4.13.2.11 DataObject* Buffer::map (const Reference & *ref*) const [inline]

Maps the provided **Reference**(p. 378) to its corresponding **DataObject**(p. 145).

Parameters:

ref The **Reference**(p. 378) to find a **DataObject**(p. 145) for.

Returns:

The **DataObject**(p. 145) for the provided **Reference**(p. 378) of null if no such **DataObject**(p. 145) was found..

4.13.2.12 const DataObject* Buffer::originalSimulation () const [inline]

Accessor for the original simulation **DataObject**(p. 145), i.e. the one that the server got initialized with.

Returns:

The original simulation **DataObject**(p. 145).

4.13.2.13 void Buffer::put (std::vector< Update * > *updates*)

Stores data about updates in the Buffer.

Called by a **Session**(p. 408) after receiving an **UpdateMessage**(p. 570) from an active client. From this point, the Buffer takes over responsibility for deallocating memory used by the updates.

Parameters:

updates A vector containing the updates.

4.13.2.14 void Buffer::put (DataObject * *d*)

Stores data about a simulation in the Buffer.

Called by a **Session**(p. 408) after receiving an InitializationMessage from an active client.

Parameters:

d The **DataObject**(p. 145) for the simulation.

4.13.2.15 void Buffer::reset ()

Resets the buffer to the same state as directly after an initialization.

Called by the **Engine**(p. 189) when told to reset the simulation.

4.13.2.16 int Buffer::resetCount () const [inline]

Accessor for the reset count.

Returns:
The reset count.

4.13.2.17 void Buffer::simTime (Time *simTime*) [inline]

Mutator for the time for which the data in the Buffer is valid.

Parameters:
simTime The time for which the data in the Buffer is valid.

4.13.2.18 Time Buffer::simTime () const [inline]

Accessor for the time for which the data in the Buffer is valid.

Returns:
The time for which the data in the Buffer is valid.

4.13.2.19 const DataObject* Buffer::simulation () const [inline]

Accessor for the simulation **DataObject**(p. 145).

Returns:
The simulation **DataObject**(p. 145).

4.13.2.20 string Buffer::simulationName () const

Returns the name of the currently initialized simulation.

Used in order to produce LoadQuery messages.

Returns:
The name of the currently initialized simulation.

4.13.2.21 void Buffer::transferUpdatesToSimulation ()

Transfers the updates from the Buffer to the simulation.

Called by the **Engine**(p. 189) when told to update the simulation.

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

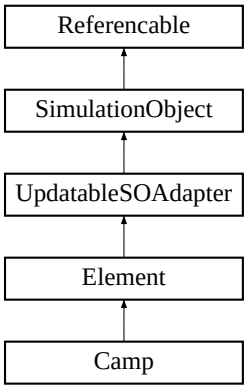
- Buffer.h
- Buffer.cpp

4.14 Camp Class Reference

Class representing a refugee camp.

`#include <Camp.h>`

Inheritance diagram for Camp::



Public Member Functions

- **Camp** (const **Reference** &ref, const **Shape** &location, int nFactions)
  Constructor.
- ~**Camp** ()
  Destructor.
- bool **present** () const
  A Camp is always present once it is created.
- double **population** (int f=0)
  Access the population of a specified **Faction**(p. 212).
- void **addPopulation** (double pop, int f=0)
  Add population of a specified **Faction**(p. 212) *to this Camp.*

Private Attributes

- double * **mPopulation**
  Population of each **Faction**(p. 212) *in this Camp.*

4.14.1 Detailed Description

Class representing a refugee camp.

Author:
Per Alexius

Date Date
2006/02/28 17:48:10

4.14.2 Constructor & Destructor Documentation

4.14.2.1 Camp::Camp (const Reference & *ref*, const Shape & *location*, int *nFactions*) [inline]

Constructor.

Parameters:
ref The **Reference**(p. 378) to this Camp.
location A **Shape**(p. 412) defining the area of this Camp.
nFactions The total number of Factions excluding the 'all' **Faction**(p. 212).

4.14.3 Member Function Documentation

4.14.3.1 void Camp::addPopulation (double *pop*, int *f* = 0) [inline]

Add population of a specified **Faction**(p. 212) to this Camp.

Parameters:
pop The population to add.
f Index of the **Faction**(p. 212) to get the population value for.

4.14.3.2 double Camp::population (int *f* = 0) [inline]

Access the population of a specified **Faction**(p. 212).

Parameters:
f Index of the **Faction**(p. 212) to get the population value for.

Returns:
The population of **Faction**(p. 212) f in this Camp

4.14.3.3 bool Camp::present () const [inline, virtual]

A Camp is always present once it is created.

Returns:
Always true.

Implements **Element** (p. 182).

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

- Camp.h

4.15 castPredicate< Base, Sub > Class Template Reference

Unary predicate for controlling if an object of type Base may be dynamic_cast to type Sub. Only works for polymorphic classes i.e. classes with at least one virtual function.

Public Member Functions

- bool **operator()** (const Base b)

  Returns true if b may be dynamic_cast to type Sub.

4.15.1 Detailed Description

template<class Base, class Sub> class castPredicate< Base, Sub >

Unary predicate for controlling if an object of type Base may be dynamic_cast to type Sub. Only works for polymorphic classes i.e. classes with at least one virtual function.

Author:
Per Alexius

Date:
2006/10/02 16:03:46

4.15.2 Member Function Documentation

4.15.2.1 template<class Base, class Sub> bool castPredicate< Base, Sub >::operator() (const Base *b*) [inline]

Returns true if b may be dynamic_cast to type Sub.

Parameters:
b Object of type Base

Returns:
True if b may be dynamic_cast to type Sub.

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

- AgencyFactory.cpp

4.16 CellGroup Class Reference

Represents a collection of cells.

`#include <CellGroup.h>`

Public Member Functions

- **CellGroup** (const **GridDataHandler** &gdh)

  Constructor.

- ~**CellGroup** ()

  Destructor.

- int **size** () const

  Accessor for the number of cells in this group.

- void **addMember** (const **GridCell** *c, double w=1.0)

  Adds a Attributes object to this group.

- const std::vector< std::pair< double, const **GridCell** * > > & **members** () const
- double **pvfGet** (ePVF pv, int f=0) const

  Accessor for values for PV:s with faction.

- double **pvGet** (ePV pv) const

  Accessor for values for PV:s without faction.

- double **pdfGet** (eDerivedF pv, int f=0) const

  Accessor for values for derived PV:s with faction.

- double **pdGet** (eDerived pv) const

  Accessor for values for derived PV:s without faction.

- double **pcfGet** (ePreCalcF pv, int f=0) const

  Accessor for values for precalculated PV:s with faction.

- double **pcGet** (ePreCalc pv) const

  Accessor for values for precalculated PV:s without faction.

- void **update** ()

  Accessor for values for stance layers.

- void **updateWeights** ()

  Sets the weight of the member cells so that a cell overlapped by n regions gets the weight 1/n.

Private Member Functions

- void **zero** ()

  Resets all aggregates to zero.

- void **handleRoundOffErrors** ()

  Handles round off errors.

Private Attributes

- const **GridDataHandler** & **mGDH**

  The GridDatahandler.

- int **mFactions**

  The number of factions.

- double ** **mPVF**

  Array of arrays of aggregated PV values for PV:s with faction.

- double * **mPV**

  Array of aggregated values for PV:s without faction.

- double ** **mPDF**

  Array of arrays of aggregated values for derived PV:s with faction.

- double * **mPD**

  Array of aggregated values for derived PV:s without faction.

- double ** **mPCF**

  Array of arrays of aggregated values for precalculated PV:s with faction.

- double * **mPC**

  Array of aggregated values for precalculated PV:s without faction.

- std::vector< std::pair< double, const **GridCell** * > > **mMembers**

  Array of aggregated values for stance layers. Vector containing pointers to all Attributes objects that should be aggregated and how to weight them.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **CellGroup** &c)

  For debugging purposes.

4.16.1 Detailed Description

Represents a collection of cells.

This class is used to keep track of groups of gridcells and to aggregate the cells' pv values - for example when subscribing to a **Region**(p. 386).

Author:
Per Alexius

Date:
2006/10/02 16:05:13

4.16.2 Constructor & Destructor Documentation

4.16.2.1 CellGroup::CellGroup (const GridDataHandler & *gdh*)

Constructor.

Parameters:
gdh The **GridDataHandler**(p. 249).

4.16.3 Member Function Documentation

4.16.3.1 void CellGroup::addMember (const GridCell * *c*, double *w* = 1.0) [inline]

Adds a Attributes object to this group.

Parameters:
a A pointer to the Attributes object to add.
w The weight.

4.16.3.2 double CellGroup::pcfGet (ePreCalcF *pv*, int *f* = 0) const [inline]

Accessor for values for precalculated PV:s with faction.

Parameters:
pv The index of the precalculated PV.
f The faction index.

Returns:
The value of the specified precalculated PV and faction.

4.16.3.3 double CellGroup::pcGet (ePreCalc *pv*) const [inline]

Accessor for values for precalculated PV:s without faction.

Parameters:
pv The index of the precalculated PV.

Returns:
The value of the specified precalculated PV.

4.16.3.4 double CellGroup::pdfGet (eDerivedF *pv*, int *f* = 0) const [inline]

Accessor for values for derived PV:s with faction.

Parameters:
pv The index of the derived PV.

f The faction index.

Returns:
The value of the specified derived PV and faction.

4.16.3.5 double CellGroup::pdGet (eDerived *pv*) const [inline]

Accessor for values for derived PV:s without faction.

Parameters:
pv The index of the derived PV.

Returns:
The value of the specified derived PV.

4.16.3.6 double CellGroup::pvfGet (ePVF *pv*, int *f* = 0) const [inline]

Accessor for values for PV:s with faction.

Parameters:
pv The index of the PV.

f The faction index.

Returns:
The value of the specified PV and faction.

4.16.3.7 double CellGroup::pvGet (ePV *pv*) const [inline]

Accessor for values for PV:s without faction.

Parameters:
pv The index of the PV.

Returns:
The value of the specified PV.

4.16.3.8 int CellGroup::size () const [inline]

Accessor for the number of cells in this group.

Returns:
The number of cells in this group.

4.16.3.9 void CellGroup::update ()

Accessor for values for stance layers.

Parameters:
pv The index of the stance layer

Returns:
The value of the specified stance layer.

4.16.4 Friends And Related Function Documentation

4.16.4.1 std::ostream& operator<< (std::ostream & *o*, const CellGroup & *c*) [friend]

For debugging purposes.

Parameters:
o The stream to write to.
c The CellGroup to print.

Returns:
The stream with the CellGroup written to it.

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

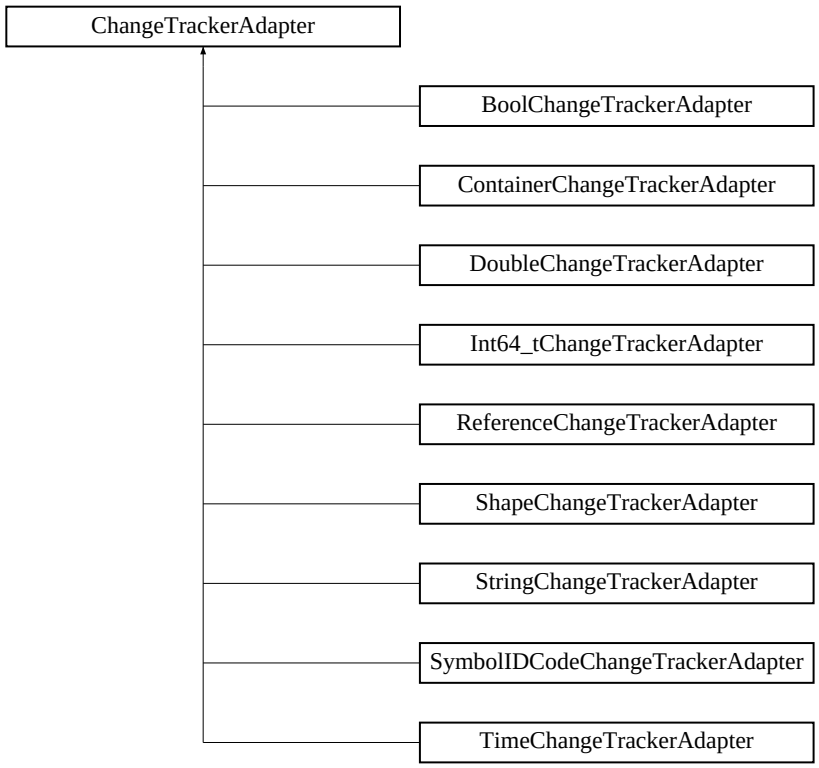
- CellGroup.h
- CellGroup.cpp

4.17 ChangeTrackerAdapter Class Reference

This is the abstract super class for all types of ChangeTrackerAdapters. A ChangeTrackerAdapter is an object that is used to keep track of changes in DataObjects. They are used in order to deliver no more than the necessary update information to the clients.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for ChangeTrackerAdapter::



Public Member Functions

- virtual bool **changed** () const =0

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 84) *function.*

- std::ostream & **toXML** (std::ostream &o)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema.

- virtual std::ostream & **toXML** (std::ostream &o, std::string indent)=0

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

4.17.1 Detailed Description

This is the abstract super class for all types of ChangeTrackerAdapters. A ChangeTrackerAdapter is an object that is used to keep track of changes in DataObjects. They are used in order to deliver no more than the necessary update information to the clients.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.17.2 Member Function Documentation

4.17.2.1 virtual bool ChangeTrackerAdapter::changed () const [pure virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 84) function.

Returns:
True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 84) function, false otherwise.

Implemented in **BoolChangeTrackerAdapter** (p. 66), **DoubleChangeTrackerAdapter** (p. 177), **Int64_tChangeTrackerAdapter** (p. 264), **ReferenceChangeTrackerAdapter** (p. 385), **ShapeChangeTrackerAdapter** (p. 419), **StringChangeTrackerAdapter** (p. 497), **TimeChangeTrackerAdapter** (p. 521), **SymbolIDCodeChangeTrackerAdapter** (p. 507), and **ContainerChangeTrackerAdapter** (p. 127).

4.17.2.2 virtual std::ostream& ChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [pure virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:
o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:
The provided ostream with the XML representation written to it.

Implemented in **BoolChangeTrackerAdapter** (p. 66), **DoubleChangeTrackerAdapter** (p. 177), **Int64_tChangeTrackerAdapter** (p. 264), **ReferenceChangeTrackerAdapter** (p. 385), **ShapeChangeTrackerAdapter** (p. 419), **StringChangeTrackerAdapter** (p. 497), **TimeChangeTrackerAdapter** (p. 521), **SymbolIDCodeChangeTrackerAdapter** (p. 507), and **ContainerChangeTrackerAdapter** (p. 128).

4.17.2.3 std::ostream& ChangeTrackerAdapter::toXML (std::ostream & *o*) [inline]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema.

Parameters:

o The ostream to write the XML representation to.

Returns:

The provided ostream with the XML representation written to it.

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

- ChangeTrackerAdapter.h

4.18 ChangeTrackerAdapterFactory Class Reference

The ChangeTrackerAdapterFactory is used to create ChangeTrackerAdapters.

`#include <ChangeTrackerAdapter.h>`

Static Public Member Functions

- **ChangeTrackerAdapter** * **createChangeTrackerAdapter** (const **Reference** &r, int64_t id)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the object refered to by the provided* **Reference**(p. 378).

4.18.1 Detailed Description

The ChangeTrackerAdapterFactory is used to create ChangeTrackerAdapters.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.18.2 Member Function Documentation

4.18.2.1 ChangeTrackerAdapter * ChangeTrackerAdapterFactory::createChangeTrackerAdapter (const Reference & *r*, int64_t *id*) [static]

Creates a **ChangeTrackerAdapter**(p. 82) for the object refered to by the provided **Reference**(p. 378).

Parameters:
r The **Reference**(p. 378) to the **DataObject**(p. 145) to adapt.

id The id of the **Session**(p. 408) the created adapter will belong to.

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

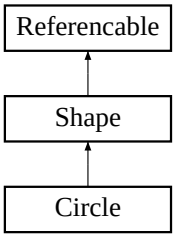
- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.19 Circle Class Reference

A class representing a Circle.

`#include <Shape.h>`

Inheritance diagram for Circle::



Public Member Functions

- **Circle** (const **LatLng** l, double r, const **Reference** &ref)
  Creates a Circle.
- **Circle** (const **LatLng** l, double r)
  Creates a Circle with no **Reference**(p. 378).
- virtual ~**Circle** ()
  Destructor.
- void **toProj** (const **Projection** &proj)
  Projects this **Shape**(p. 412) *using the specified* **Projection**(p. 348).
- void **toCoord** (const **Projection** &proj)
  Transforms this **Shape**(p. 412) *to lat lng coordinate using the provided projection.*
- void **cells** (const **BasicGrid** &g, std::list< **GridPos** > &outCells) const
  Returns a list containing pointers to all cells covered by this **Shape**(p. 412).
- **LatLng cenCoord** () const
  Returns the center coordinate in lat lng of this **Shape**(p. 412).
- **ProjCoord cenProj** () const
  Returns the center coordinate in projection space of this **Shape**(p. 412).
- void **boundingBox** (double &t, double &l, double &b, double &r) const
  Gets the bounding box of this **Shape**(p. 412).
- double **area** () const
  Returns the area of this **Shape**(p. 412).
- void **move** (double dx, double dy)
  Moves this **Shape**(p. 412) *relative to itself.*

- void **move** (**LatLng** newPos)
  Moves this **Shape**(p. 412) *to a new position.*
- **Shape** * **clone** () const
  Creates a deep copy of this **Shape**(p. 412).
- const std::string **type** () const
  Returns the stratmas protocol type of this shape.
- double **radius** () const
  Accessor for the raduis of this Circle.
- std::ostream & **toXML** (std::ostream &o, std::string indent) const
  Writes an XML representation of this object to the provided stream with nice indentation.

Protected Attributes

- **LatLng mCenter**
  The center of this Circle.
- double **mRadius**
  The radius of this Circle.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Circle** &c)
  For debugging purposes.

4.19.1 Detailed Description

A class representing a Circle.

Author:
Per Alexius

Date:
2006/07/19 07:04:39

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Circle::Circle (const LatLng *l*, double *r*, const Reference & *ref*) [inline]

Creates a Circle.

Parameters:
l The center position of the Circle.

r The radius of the Circle.

ref The **Reference**(p. 378) to the Circle.

4.19.2.2 Circle::Circle (const LatLng *l*, double *r*)

Creates a Circle with no **Reference**(p. 378).

Parameters:

l The center position of the Circle.

r The radius of the Circle.

4.19.3 Member Function Documentation

4.19.3.1 double Circle::area () const [inline, virtual]

Returns the area of this **Shape**(p. 412).

Returns:

The area of this **Shape**(p. 412).

Implements **Shape** (p. 414).

4.19.3.2 void Circle::boundingBox (double & *t*, double & *l*, double & *b*, double & *r*) const [virtual]

Gets the bounding box of this **Shape**(p. 412).

Parameters:

t Top coordinate of this Shape's boundingbox.

l Left coordinate of this Shape's boundingbox.

b Bottom coordinate of this Shape's boundingbox.

r Right coordinate of this Shape's boundingbox.

Implements **Shape** (p. 414).

4.19.3.3 void Circle::cells (const BasicGrid & *g*, std::list< GridPos > & *outCells*) const [virtual]

Returns a list containing pointers to all cells covered by this **Shape**(p. 412).

Parameters:

g A reference to the **Grid**(p. 227).

outCells A list that on return contains pointers to all cells covered by this **Shape**(p. 412).

Implements **Shape** (p. 414).

4.19.3.4 LatLng Circle::cenCoord () const [inline, virtual]

Returns the center coordinate in lat lng of this **Shape**(p. 412).

Returns:
The center coordinate of this **Shape**(p. 412).

Implements **Shape** (p. 415).

4.19.3.5 ProjCoord Circle::cenProj () const [inline, virtual]

Returns the center coordinate in projection space of this **Shape**(p. 412).

Returns:
The center coordinate of this **Shape**(p. 412).

Implements **Shape** (p. 415).

4.19.3.6 Shape* Circle::clone () const [inline, virtual]

Creates a deep copy of this **Shape**(p. 412).

Returns:
A newly allocated copy of this **Shape**(p. 412).

Implements **Shape** (p. 416).

4.19.3.7 void Circle::move (LatLng *newPos*) [inline, virtual]

Moves this **Shape**(p. 412) to a new position.

Parameters:
newPos The position to move to.

Implements **Shape** (p. 416).

4.19.3.8 void Circle::move (double *dx*, double *dy*) [inline, virtual]

Moves this **Shape**(p. 412) relative to itself.

Parameters:
dx The movement in x-direction in degrees longitude

dy The movement in y-direction in degrees latitude

Implements **Shape** (p. 416).

4.19.3.9 double Circle::radius () const [inline]

Accessor for the raduis of this Circle.

Returns:
The raduis of this Circle.

4.19.3.10 void Circle::toCoord (const Projection & *proj*) [inline, virtual]

Transforms this **Shape**(p. 412) to lat lng coordinate using the provided projection.

Parameters:
proj The projection to use.

Implements **Shape** (p. 416).

4.19.3.11 void Circle::toProj (const Projection & *proj*) [inline, virtual]

Projects this **Shape**(p. 412) using the specified **Projection**(p. 348).

Parameters:
proj The projection to use.

Implements **Shape** (p. 416).

4.19.3.12 ostream & Circle::toXML (std::ostream & *o*, std::string *indent*) const [virtual]

Writes an XML representation of this object to the provided stream with nice indentation.

Parameters:
o The stream to write to.
indent Indentation string.

Returns:
The stream with the xml representation written to it.

Implements **Shape** (p. 417).

4.19.3.13 const std::string Circle::type () const [inline, virtual]

Returns the stratmas protocol type of this shape.

Returns:
The stratmas protocol type of this shape.

Implements **Shape** (p. 417).

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

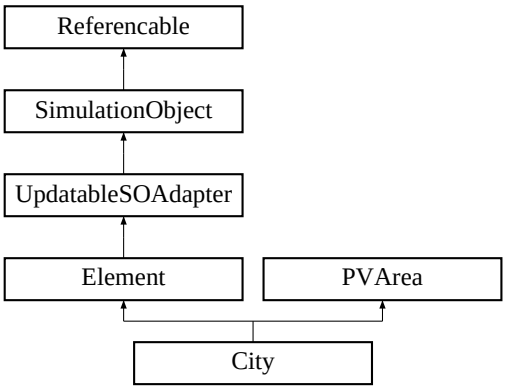
- Shape.h
- Shape.cpp

4.20 City Class Reference

City is the class containing Stratmas' representation of a City, or more general - a population instance.

`#include <City.h>`

Inheritance diagram for City::



Public Member Functions

- **City** (const **DataObject** &d)

  Creates a City from the provided **DataObject**(p. 145).

- bool **present** () const

  A City is always present.

- virtual void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- double **population** (int f=0) const

  Gets the population for the **Faction**(p. 212) *with the provided index.*

- double **population** (const **Faction** &f) const

  Gets the population for the specified **Faction**(p. 212).

- const **Shape** & **area** () const

  Accessor for the **Shape**(p. 412) *the modifications refer to.*

- const **Distribution** & **distribution** () const

  Accessor for the **Distribution**(p. 173) *of the modifications over the area.*

Private Attributes

- int **mFactions**

  The total number of Factions in the simulation.

- std::map< const **Reference** *, double > **mPop**

  Maps a faction **Reference**(p. 378) *to the number of inhabitants for that faction.*

Friends

- std::ostream & **operator**<< (std::ostream &o, const **City** &c)

  For debugging purposes.

4.20.1 Detailed Description

City is the class containing Stratmas' representation of a City, or more general - a population instance.

A City has an area, a population from one or more Factions and a population **Distribution**(p. 173). Currently all cities have the same kind of **Distribution**(p. 173) - the StratmasCityDistribution that represents the way older versions of Stratmas distributed population from cities.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 City::City (const DataObject & *d*)

Creates a City from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this City from.

4.20.3 Member Function Documentation

4.20.3.1 const Shape& City::area () const [inline, virtual]

Accessor for the **Shape**(p. 412) the modifications refer to.

Returns:

The **Shape**(p. 412).

Implements **PVArea** (p. 355).

4.20.3.2 const Distribution& City::distribution () const [inline, virtual]

Accessor for the **Distribution**(p. 173) of the modifications over the area.

Returns:

The **Distribution**(p. 173).

Implements **PVArea** (p. 356).

4.20.3.3 virtual void City::extract (Buffer & *b*) const [inline, virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Reimplemented from **Element** (p. 182).

4.20.3.4 double City::population (const Faction & *f*) const

Gets the population for the specified **Faction**(p. 212).

Parameters:
f The faction.

Returns:
The number of inhabitants for the specifid **Faction**(p. 212).

4.20.3.5 double City::population (int *f* = 0) const

Gets the population for the **Faction**(p. 212) with the provided index.

Parameters:
f The faction index.

Returns:
The number of inhabitants for the specifid **Faction**(p. 212).

4.20.3.6 bool City::present () const [inline, virtual]

A City is always present.

Returns:
Always true.

Implements **Element** (p. 182).

4.20.4 Friends And Related Function Documentation

4.20.4.1 std::ostream& operator<< (std::ostream & *o*, const City & *c*) [friend]

For debugging purposes.

Parameters:
o The ostream to write to.

c The City to write.

Returns:
The provided ostream with the City written to it.

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

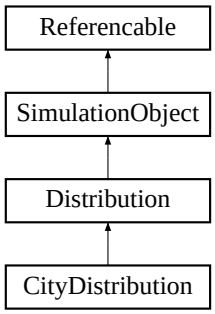
- City.h
- City.cpp

4.21 CityDistribution Class Reference

The distibution used to spread population from cities.

`#include <Distribution.h>`

Inheritance diagram for CityDistribution::



Public Member Functions

- **CityDistribution** (const **DataObject** &d)

  Creates a CityDistribution from the specified **DataObject**(p. 145).

- double **f** (double x) const

  Gets the value of the distribution at distance x.

- void **update** (const **Update** &u)

  Updates this object.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Private Attributes

- double **mSigma**

  The diffusion measure.

- double **mK**

  Distribution(p. 173) *constant.*

4.21.1 Detailed Description

The distibution used to spread population from cities.

Author:
Per Alexius

Date:
2006/04/21 15:54:49

4.21.2 Constructor & Destructor Documentation

4.21.2.1 CityDistribution::CityDistribution (const DataObject & *d*)

Creates a CityDistribution from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.21.3 Member Function Documentation

4.21.3.1 void CityDistribution::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.21.3.2 double CityDistribution::f (double *x*) const [inline, virtual]

Gets the value of the distribution at distance x.

Parameters:
x The distance.

Returns:
The value of the distribution at distance x.

Implements **Distribution** (p. 175).

4.21.3.3 void CityDistribution::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Distribution** (p. 175).

4.21.3.4 void CityDistribution::update (const Update & *u*) [virtual]

Updates this object.

Parameters:

u The **Update**(p. 567) to update this object with.

Reimplemented from **Distribution** (p. 175).

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

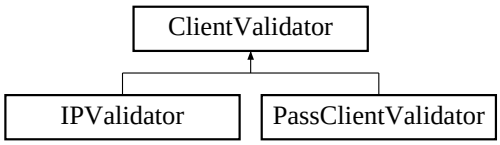
- Distribution.h
- Distribution.cpp

4.22 ClientValidator Class Reference

Class that validetes if a client may connect.

`#include <ClientValidator.h>`

Inheritance diagram for ClientValidator::



Public Member Functions

- virtual bool **isValidClient** (const **Socket** *socket)=0

4.22.1 Detailed Description

Class that validetes if a client may connect.

Author:
Daniel Ahlin

Date:
2006/07/21 13:35:29

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

- ClientValidator.h

4.23 ClusterSet Class Reference

Performs clustering on a multivariate dataset. Each cluster is an ellipse in general position.

`#include <ClusterSet.h>`

Public Member Functions

- **ClusterSet** (int max)
- ~**ClusterSet** ()
- int **FitToData** (int n, int *index, const double *const xp, const double *const yp, double *wp)

  Cluster this weighted data set, using the k-means algorithm.

- void **MakeSigmaBoxes** (double k)

  Undocumented by Loren Cobb.

Public Attributes

- int **mNumClusters**

  Number of clusters found.

- int **mMaxClusters**

  Maximum number of clusters to look for.

- **Ellipse** * **mCluster**

  The array of cluster ellipses.

4.23.1 Detailed Description

Performs clustering on a multivariate dataset. Each cluster is an ellipse in general position.

This class is almost identical to its counterpart in older versions of Stratmas. Some name changes have been made in order to match naming conventions in other source code files.

Author:
Loren Cobb - Modified by Per Alexius

Date:
2005/06/03 17:14:03

4.23.2 Constructor & Destructor Documentation

4.23.2.1 ClusterSet::ClusterSet (int *n*)

Constructor.

Parameters:
n The maximal number of clusters.

4.23.2.2 ClusterSet::~ClusterSet ()

Destructor.

4.23.3 Member Function Documentation

4.23.3.1 int ClusterSet::FitToData (int *n*, int * *index*, const double *const *xp*, const double *const *yp*, double * *wp*)

Cluster this weighted data set, using the k-means algorithm.

Parameters:

n Length of data vectors

index Vector of cluster ids

xp Vector of x-coords

yp Vector of y-coords

wp Vector of weights

Returns:

The number of clusters found.

4.23.3.2 void ClusterSet::MakeSigmaBoxes (double *k*)

Undocumented by Loren Cobb.

Parameters:

k Undocumented by Loren Cobb.

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

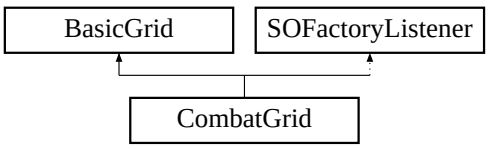
- ClusterSet.h
- ClusterSet.cpp

4.24 CombatGrid Class Reference

This class controlls most of the grid related combat activities such as finding out which units that overlaps etc.

`#include <CombatGrid.h>`

Inheritance diagram for CombatGrid::



Public Member Functions

- **CombatGrid** (**Map** &amap, **Grid** &grid, std::vector< **Unit** * > &forceVec, const std::vector< **Faction** * > &facVec)

  Creats a CombatGrid for the specified map and forces.

- ~**CombatGrid** ()

  Destructor.

- void **objectAdded** (const **Reference** &ref, int64_t initiator)

  Called when an object has been added by the **SOFactory**(p. 436).

- void **objectRemoved** (const **Reference** &ref, int64_t initiator)

  Called when an object has been removed by the **SOFactory**(p. 436).

- int **layers** () const

  Accessor for the number of layers.

- int **blueLayer** () const

  Accessor for the blue layer index.

- int **casualtySumLayer** () const

  Accessor for the casualty sum layer index.

- double **value** (int actInd, int layer) const

  Gets the value for the specified cell and layer.

- double **value** (int actInd, const std::string &layer) const

  Gets the value for the specified cell and layer.

- double **value** (**GridPos** gp, int layer) const

  Gets the value for the specified cell and layer.

- void **add** (int actInd, int layer, double val) const

  Adds a value to the specified cell and layer.

- double * **aggregate** (const std::list< **GridPos** > &pos, double *&outAgg) const

  Aggregates (by summing) values for all layers for the specified cells.

- double * **aggregate** (const **Shape** &region, double *&outAgg) const

  Aggregates (by summing) values for all layers for the cells covered by the provided **Shape**(p. 412).

- void **reset** (std::vector< **Unit** * > &forceVec)

  Resets the CombatGrid.

- void **unitsToGrid** ()

  Calculates the personnel in each cell for all units.

- void **setUpBattleField** ()

  Setup function that should be called at the beginning of each timestep, before the units act.

- void **registerCombat** ()

  Registration function that should be called at the end of each timestep, after the units have acted.

- const std::map< std::string, int > & **nameToIndexMap** () const

  Accessor for the mNameToIndex map.

- std::string **indexToName** (int i) const

  Maps the name of a layer to its index.

- int **nameToIndex** (const std::string &name) const

  Maps the index of a layer to its name.

Static Public Attributes

- const char * **kCasualtyStr** = " casualties"

Private Member Functions

- void **reset** (bool includeNonResetables=false)

  Resets grid layers.

- void **unitToGrid** (**Unit** &u)

  Adds the personnel of the provided unit to the cells it covers using the deployment **Distribution**(p. 173).

- void **markPresence** (**Unit** &u)

  Marks the presence of the provided **Unit**(p. 546) *by creating a* **PresenceObject**(p. 338) *for every cell the* **Unit**(p. 546) *overlaps.*

- void **markPresence** ()

  Create PresenceObjects for all cells overlapped by any unit.

Private Attributes

- std::map< std::string, int > **mNameToIndex**

  Maps the name of a layer to its index.

- std::vector< std::string > **mIndexToName**

  Maps the index of a layer to its name.

- bool * **mResetableLayer**

  An array that contains a bool value for each layer indicating whether or not the layer should be reset between two consecutive timesteps.

- **Grid** & **mGrid**

  Reference(p. 378) *to the* **Grid**(p. 227).

- int **mNumLayers**

  The number of layers in the CombatGrid.

- int **mSumLayerIndex**

  Index of the layer storing the sum of all units' personnel.

- int **mCasualtySumLayerIndex**

  Index of the layer storing the sum of all units' personnel.

- int **mBlueLayerIndex**

  The index of the layer storing the sum of all blue forces' personnel. A blue force is a force that has 'F' as the second letter in its symbol id code.

- int **mInsurgentLayerIndex**

  Index of the layer storing the number of insurgent casualties.

- std::vector< **Unit** * > & **mForces**

  A reference to the scenario's force vector.

- double ** **mGridData**

  Two dimensional array storing the actual values in the grid.

- std::set< **PresenceObject** *, **lessPresenceObjectPointer** > **mPresence**

  The set of PresenceObjects marking the presence of all units during each timestep.

- **PresenceObjectAllocator** **mPOA**

  Memory handler for PresenceObjects.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **CombatGrid** &c)

  For debugging purposes.

4.24.1 Detailed Description

This class controlls most of the grid related combat activities such as finding out which units that overlaps etc.

Author:
Per Alexius

Date:
2006/09/04 14:34:39

4.24.2 Constructor & Destructor Documentation

4.24.2.1 CombatGrid::CombatGrid (Map & *amap*, Grid & *grid*, std::vector< Unit * > & *forceVec*, const std::vector< Faction * > & *facVec*)

Creats a CombatGrid for the specified map and forces.

Parameters:
amap The map.

grid The **Grid**(p. 227).

forceVec A vector with the top units in all forces.

4.24.3 Member Function Documentation

4.24.3.1 void CombatGrid::add (int *actInd*, int *layer*, double *val*) const [inline]

Adds a value to the specified cell and layer.

Parameters:
actInd The index of the cell in the active array.

layer The index of the layer.

val The value to add.

4.24.3.2 double * CombatGrid::aggregate (const Shape & *region*, double *& *outAgg*) const

Aggregates (by summing) values for all layers for the cells covered by the provided **Shape**(p. 412).

Parameters:
region A **Shape**(p. 412) specifying the region over which to aggregate.

outAgg An array that must be of at least mNumLayers lenght in which the aggregated values will be placed on return.

Returns:
The outAgg array.

4.24.3.3 double * CombatGrid::aggregate (const std::list< GridPos > & *pos*, double *& *outAgg*) const

Aggregates (by summing) values for all layers for the specified cells.

Parameters:

pos A list of **GridPos**(p. 256) specifying the cells to aggregate.

outAgg An array that must be of at least mNumLayers lenght in which the aggregated values will be placed on return.

Returns:

The outAgg array.

4.24.3.4 int CombatGrid::blueLayer () const [inline]

Accessor for the blue layer index.

Returns:

The blue layer index.

4.24.3.5 int CombatGrid::casualtySumLayer () const [inline]

Accessor for the casualty sum layer index.

Returns:

The casualty sum layer index.

4.24.3.6 std::string CombatGrid::indexToName (int *i*) const [inline]

Maps the name of a layer to its index.

Parameters:

i The layer index.

Returns:

The name of the specified layer.

4.24.3.7 int CombatGrid::layers () const [inline]

Accessor for the number of layers.

Returns:

The number of layers.

4.24.3.8 void CombatGrid::markPresence () [private]

Create PresenceObjects for all cells overlapped by any unit.

For each cell overlapped by a unit we will also create a **PresenceObject**(p. 338) for the insurgents in that cell.

4.24.3.9 void CombatGrid::markPresence (Unit & *u*) [private]

Marks the presence of the provided **Unit**(p. 546) by creating a **PresenceObject**(p. 338) for every cell the **Unit**(p. 546) overlaps.

Parameters:
u The unit to mark presence for.

4.24.3.10 int CombatGrid::nameToIndex (const std::string & *name*) const [inline]

Maps the index of a layer to its name.

Parameters:
name The layer name.

Returns:
The index of the specified layer.

4.24.3.11 const std::map<std::string, int>& CombatGrid::nameToIndexMap () const [inline]

Accessor for the mNameToIndex map.

Returns:
The mNameToIndex map.

4.24.3.12 void CombatGrid::objectAdded (const Reference & *ref*, int64_t *initiator*) [virtual]

Called when an object has been added by the **SOFactory**(p. 436).

Both the **SimulationObject**(p. 429) and the corresponding **DataObject**(p. 145) exists and are registered when this call occurs.

Parameters:
ref The **Reference**(p. 378) to the object that was added.

initiator The id of the initiator of the event.

Implements **SOFactoryListener** (p. 450).

4.24.3.13 void CombatGrid::objectRemoved (const Reference & *ref*, int64_t *initiator*) [virtual]

Called when an object has been removed by the **SOFactory**(p. 436).

When this function is called the **SimulationObject**(p. 429) is already deleted and deregistered. The corresponding **DataObject**(p. 145) does still exist and is still registered.

Parameters:

ref The **Reference**(p. 378) to the object that is removed

initiator The id of the initiator of the event.

Implements **SOFactoryListener** (p. 451).

4.24.3.14 void CombatGrid::registerCombat ()

Registration function that should be called at the end of each timestep, after the units have acted.

Handles insurgent combat and registers casualties etc.

4.24.3.15 void CombatGrid::reset (std::vector< Unit * > & *forceVec*)

Resets the CombatGrid.

Called only when resetting a **Scenario**(p. 396).

Parameters:

forceVec A vector with all top units in the scenario.

4.24.3.16 void CombatGrid::reset (bool *includeNonResetables* = false) [private]

Resets grid layers.

May reset all layers (as when reseting the whole CombatGrid) or only the resetable layers (as in each timestep).

Parameters:

includeNonResetables Indicates whether the 'non resetable' layers should be reset.

4.24.3.17 void CombatGrid::setUpBattleField ()

Setup function that should be called at the beginning of each timestep, before the units act.

Marks the presence of all units and registers which units that will fight each other etc.

4.24.3.18 void CombatGrid::unitToGrid (Unit & *u*) [private]

Adds the personnel of the provided unit to the cells it covers using the deployment **Distribution**(p. 173).

Parameters:

u The unit which personnel should be added to the grid.

4.24.3.19 double CombatGrid::value (GridPos *gp*, int *layer*) const [inline]

Gets the value for the specified cell and layer.

Parameters:

gp The **GridPos**(p. 256) marking the cell.

layer The index of the layer.

Returns:

The value for the specified cell and layer.

4.24.3.20 double CombatGrid::value (int *actInd*, const std::string & *layer*) const [inline]

Gets the value for the specified cell and layer.

Parameters:

actInd The index of the cell in the active array.

layer The name of the layer.

Returns:

The value for the specified cell and layer.

4.24.3.21 double CombatGrid::value (int *actInd*, int *layer*) const [inline]

Gets the value for the specified cell and layer.

Parameters:

actInd The index of the cell in the active array.

layer The index of the layer.

Returns:

The value for the specified cell and layer.

4.24.4 Friends And Related Function Documentation

4.24.4.1 std::ostream& operator<< (std::ostream & *o*, const CombatGrid & *c*) [friend]

For debugging purposes.

Parameters:

o The stream to write to.

c The CombatGrid to print.

Returns:

The stream with the CombatGrid written to it.

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

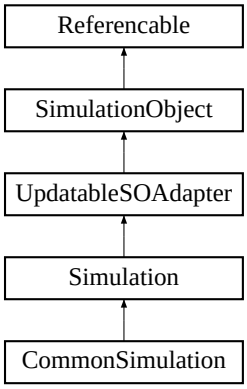
- CombatGrid.h
- CombatGrid.cpp

4.25 CommonSimulation Class Reference

Represents the common simulation.

`#include <Simulation.h>`

Inheritance diagram for CommonSimulation::



Public Member Functions

- **CommonSimulation** (const **DataObject** &d)
  Creates a CommonSimulation from the provided data object.

4.25.1 Detailed Description

Represents the common simulation.

Author:
Per Alexius

Date:
2006/10/02 16:05:16

4.25.2 Constructor & Destructor Documentation

4.25.2.1 CommonSimulation::CommonSimulation (const DataObject & *d*) [inline]

Creates a CommonSimulation from the provided data object.

Parameters:
d The data object to create this object from.

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

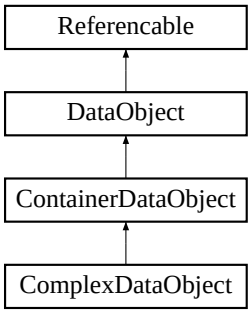
- Simulation.h

4.26 ComplexDataObject Class Reference

ComplexDataObjects represent complex objects in the Stratmas xml schema except ValueType descendants.

`#include <DataObjectImpl.h>`

Inheritance diagram for ComplexDataObject::



Public Member Functions

- **ComplexDataObject** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a ComplexDataObject in the provided scope from the provided DOMElement.

- **ComplexDataObject** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- void **orderPreservingAdd** (**DataObject** *o)

  Adds a child to this **ContainerDataObject**(p. 129) *preserving the order as specified in the Stratmas xml schemas.*

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

Protected Member Functions

- **ComplexDataObject** (const **ComplexDataObject** &c)

  Copy constructor.

4.26.1 Detailed Description

ComplexDataObjects represent complex objects in the Stratmas xml schema except ValueType descendants.

Author:
Per Alexius

Date Date

2006/03/27 09:43:40

4.26.2 Constructor & Destructor Documentation

4.26.2.1 ComplexDataObject::ComplexDataObject (const ComplexDataObject & *c*) [inline, protected]

Copy constructor.

Parameters:

c The **DataObject**(p. 145) to copy.

4.26.2.2 ComplexDataObject::ComplexDataObject (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a ComplexDataObject in the provided scope from the provided DOMElement.

This constructor goes through its Type's sub Decalarations and tries to create a child for each such **Declaration**(p. 160).

Parameters:

scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.26.2.3 ComplexDataObject::ComplexDataObject (const Reference & *ref*, const Type & *type*)

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:

ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.26.3 Member Function Documentation

4.26.3.1 DataObject* ComplexDataObject::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.26.3.2 void ComplexDataObject::orderPreservingAdd (DataObject * *o*)

Adds a child to this **ContainerDataObject**(p. 129) preserving the order as specified in the Stratmas xml schemas.

Parameters:

o The child to add.

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

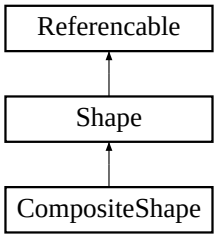
- DataObjectImpl.h
- DataObjectImpl.cpp

4.27 CompositeShape Class Reference

A class representing a CompositeShape.

`#include <Shape.h>`

Inheritance diagram for CompositeShape::



Public Member Functions

- **CompositeShape** (const **Reference** &ref)
  Creates an empty CompositeShape.
- ~**CompositeShape** ()
- void **addShape** (**Shape** *s)
  Adds a **Shape**(p. 412) *to this CompositeShape.*
- int **parts** () const
  Gets the number of Shapes in this CompositeShape.
- void **toProj** (const **Projection** &proj)
  Projects this **Shape**(p. 412) *using the specified* **Projection**(p. 348).
- void **toCoord** (const **Projection** &proj)
  Transforms this **Shape**(p. 412) *to lat lng coordinate using the provided projection.*
- void **cells** (const **BasicGrid** &g, std::list< **GridPos** > &outCells) const
  Returns a list containing pointers to all cells covered by this **Shape**(p. 412).
- **LatLng cenCoord** () const
  Returns the center coordinate in lat lng of this **Shape**(p. 412).
- **ProjCoord cenProj** () const
  Returns the center coordinate in projection space of this **Shape**(p. 412).
- void **boundingBox** (double &t, double &l, double &b, double &r) const
  Gets the bounding box of this **Shape**(p. 412).
- double **area** () const
  Returns the area of this **Shape**(p. 412). *Not yet implemented for CompositeShape.*
- void **move** (double dx, double dy)

Moves this **Shape**(p. 412) *relative to itself.*

- void **move** (**LatLng** newPos)

  Moves this **Shape**(p. 412) *to a new position.*

- **Shape** * **clone** () const

  Creates a deep copy of this **Shape**(p. 412).

- const std::string **type** () const

  Returns the stratmas protocol type of this shape.

- std::ostream & **toXML** (std::ostream &o, std::string indent) const

  Writes an XML representation of this object to the provided stream with nice indentation.

- void **getFlattened** (std::vector< **Shape** * > &shapes) const

  Gets a vector containing all Shapes this CompositeShape and its subshapes contains.

- const **Shape** * **getRegionForPoint** (const **ProjCoord** &p) const

  Finds out in which subshape the specified point is located.

- **Shape** * **getPart** (const **Reference** &toFind) const

  Tries to find a shape with the given reference in this CompositeShape.

Private Attributes

- bool **mCenterCalculated**

  True if the center of this CompositeShape is calculated.

- **LatLng mCenter**

  The center of this CompositeShape.

- std::map< std::string, **Shape** * > **mShapes**

  A map containing all subshapes this Composite **Shape**(p. 412) *contains.*

4.27.1 Detailed Description

A class representing a CompositeShape.

Author:
Per Alexius

Date **Date**
2006/07/19 07:04:39

4.27.2 Constructor & Destructor Documentation

4.27.2.1 CompositeShape::CompositeShape (const Reference & *ref*) [inline]

Creates an empty CompositeShape.

Parameters:
ref The **Reference**(p. 378) to the **Circle**(p. 86).

4.27.2.2 CompositeShape::~CompositeShape ()

Destructor.

4.27.3 Member Function Documentation

4.27.3.1 void CompositeShape::addShape (Shape * *s*)

Adds a **Shape**(p. 412) to this CompositeShape.

Parameters:
s The **Shape**(p. 412) to add.

4.27.3.2 double CompositeShape::area () const [inline, virtual]

Returns the area of this **Shape**(p. 412). Not yet implemented for CompositeShape.

Returns:
The area of this **Shape**(p. 412).

Implements **Shape** (p. 414).

4.27.3.3 void CompositeShape::boundingBox (double & *t*, double & *l*, double & *b*, double & *r*) const [virtual]

Gets the bounding box of this **Shape**(p. 412).

Parameters:
t Top coordinate of this Shape's boundingbox.

l Left coordinate of this Shape's boundingbox.

b Bottom coordinate of this Shape's boundingbox.

r Right coordinate of this Shape's boundingbox.

Implements **Shape** (p. 414).

4.27.3.4 void CompositeShape::cells (const BasicGrid & *g*, std::list< GridPos > & *outCells*) const [virtual]

Returns a list containing pointers to all cells covered by this **Shape**(p. 412).

Parameters:

g A reference to the **Grid**(p. 227).

outCells A list that on return contains pointers to all cells covered by this **Shape**(p. 412).

Implements **Shape** (p. 414).

4.27.3.5 LatLng CompositeShape::cenCoord () const [virtual]

Returns the center coordinate in lat lng of this **Shape**(p. 412).

Returns:

The center coordinate of this **Shape**(p. 412).

Implements **Shape** (p. 415).

4.27.3.6 ProjCoord CompositeShape::cenProj () const [inline, virtual]

Returns the center coordinate in projection space of this **Shape**(p. 412).

Returns:

The center coordinate of this **Shape**(p. 412).

Implements **Shape** (p. 415).

4.27.3.7 Shape * CompositeShape::clone () const [virtual]

Creates a deep copy of this **Shape**(p. 412).

Returns:

A newly allocated copy of this **Shape**(p. 412).

Implements **Shape** (p. 416).

4.27.3.8 void CompositeShape::getFlattened (std::vector< Shape * > & *shapes*) const

Gets a vector containing all Shapes this CompositeShape and its subshapes contains.

Returns:

A vector containing pointers to all Shapes this CompositeShape and its subshapes contains.

4.27.3.9 Shape * CompositeShape::getPart (const Reference & *toFind*) const

Tries to find a shape with the given reference in this CompositeShape.

Parameters:
toFind The reference to the shape to look for.

Returns:
The found shape if successful, 0 otherwise.

4.27.3.10 const Shape * CompositeShape::getRegionForPoint (const ProjCoord & *p*) const

Finds out in which subshape the specified point is located.

Parameters:
p The point.

Returns:
The first found subschape that contains the point, or null if no such subshape could be found.

4.27.3.11 void CompositeShape::move (LatLng *newPos*) [virtual]

Moves this **Shape**(p. 412) to a new position.

Parameters:
newPos The position to move to.

Implements **Shape** (p. 416).

4.27.3.12 void CompositeShape::move (double *dx*, double *dy*) [virtual]

Moves this **Shape**(p. 412) relative to itself.

Parameters:
dx The movement in x-direction in degrees longitude
dy The movement in y-direction in degrees latitude

Implements **Shape** (p. 416).

4.27.3.13 int CompositeShape::parts () const [inline]

Gets the number of Shapes in this CompositeShape.

Returns:
The number of Shapes in this CompositeShape.

4.27.3.14 void CompositeShape::toCoord (const Projection & *proj*) [virtual]

Transforms this **Shape**(p. 412) to lat lng coordinate using the provided projection.

Parameters:
proj The projection to use.

Implements **Shape** (p. 416).

4.27.3.15 void CompositeShape::toProj (const Projection & *proj*) [virtual]

Projects this **Shape**(p. 412) using the specified **Projection**(p. 348).

Parameters:
proj The projection to use.

Implements **Shape** (p. 416).

4.27.3.16 ostream & CompositeShape::toXML (std::ostream & *o*, std::string *indent*) const [virtual]

Writes an XML representation of this object to the provided stream with nice indentation.

Parameters:
o The stream to write to.
indent Indentation string.

Returns:
The stream with the xml representation written to it.

Implements **Shape** (p. 417).

4.27.3.17 const std::string CompositeShape::type () const [inline, virtual]

Returns the stratmas protocol type of this shape.

Returns:
The stratmas protocol type of this shape.

Implements **Shape** (p. 417).

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

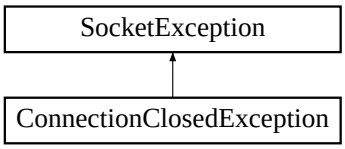
- Shape.h
- Shape.cpp

4.28 ConnectionClosedException Class Reference

Exception used by **Socket**(p. 432) class when connection is closed by client.

`#include <SocketException.h>`

Inheritance diagram for ConnectionClosedException::



Public Member Functions

- **ConnectionClosedException** ()
  Constructor.

4.28.1 Detailed Description

Exception used by **Socket**(p. 432) class when connection is closed by client.

Author:
Per Alexius

Date **Date**
2006/07/21 13:35:29

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

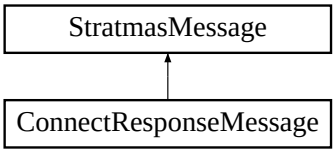
- SocketException.h

4.29 ConnectResponseMessage Class Reference

Class representing the ConnectResponseMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for ConnectResponseMessage::



Public Member Functions

- **ConnectResponseMessage** (bool active)

  Constructor.

- void **toXML** (std::ostream &o) const

  Produces the XML representation of this message.

Private Attributes

- bool **mActive**

  True if this message is sent to an active client.

4.29.1 Detailed Description

Class representing the ConnectResponseMessage.

Author:
Per Alexius

Date:
2006/03/06 14:23:12

4.29.2 Constructor & Destructor Documentation

4.29.2.1 ConnectResponseMessage::ConnectResponseMessage (bool *active*) [inline]

Constructor.

Parameters:
active Indicates whether the client receiving the message is active or not.

4.29.3 Member Function Documentation

4.29.3.1 void ConnectResponseMessage::toXML (std::ostream & *o*) const [virtual]

Produces the XML representation of this message.

Parameters:

o The stream to which the message is written

Implements **StratmasMessage** (p. 473).

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

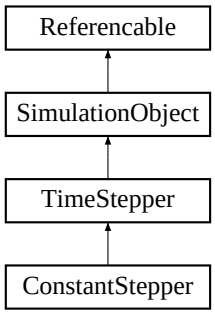
- StratmasMessage.h
- StratmasMessage.cpp

4.30 ConstantStepper Class Reference

A **TimeStepper**(p. 522) that takes timesteps of constant length.

`#include <TimeStepper.h>`

Inheritance diagram for ConstantStepper::



Public Member Functions

- **ConstantStepper** (const **DataObject** &)

  Creates a ConstantStepper from the provided **DataObject**(p. 145).

- void **update** (const **Update** &u)

  Updates this object.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- **Time dt** ()

  Get the length of the timestep.

Private Attributes

- **Time mDt**

  Length of the timestep.

4.30.1 Detailed Description

A **TimeStepper**(p. 522) that takes timesteps of constant length.

Author:
Per Alexius

Date Date
2006/03/06 14:23:13

4.30.2 Constructor & Destructor Documentation

4.30.2.1 ConstantStepper::ConstantStepper (const DataObject & *d*)

Creates a ConstantStepper from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this ConstantStepper from.

4.30.3 Member Function Documentation

4.30.3.1 Time ConstantStepper::dt () [inline, virtual]

Get the length of the timestep.

Returns:
The length of the timestep.

Implements **TimeStepper** (p. 522).

4.30.3.2 void ConstantStepper::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.30.3.3 void ConstantStepper::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

4.30.3.4 void ConstantStepper::update (const Update & *u*) [virtual]

Updates this object.

Parameters:
u The **Update**(p. 567) to update this object with.

Implements **SimulationObject** (p. 431).

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

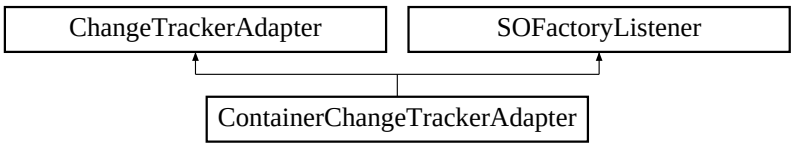
- TimeStepper.h
- TimeStepper.cpp

4.31 ContainerChangeTrackerAdapter Class Reference

The ContainerChangeTrackerAdapter keeps track of changes in StratmasContainer objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for ContainerChangeTrackerAdapter::



Public Member Functions

- **ContainerChangeTrackerAdapter** (const **Reference** &ref, int64_t id)

  Creates a ContainerChangeTrackerAdapter for the **DataObject**(p. 145) *the provided* **Reference**(p. 378) *refers to..*

- void **objectAdded** (const **Reference** &ref, int64_t initiator)

  Notifies this change tracker that an object has been added.

- void **objectRemoved** (const **Reference** &ref, int64_t initiator)

  Notifies this change tracker that an object has been removed. See **objectAdded()**(p. 127) *for more information.*

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 128) *function.*

- virtual std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Member Functions

- bool **addable** (const **Reference** &ref)

  Checks if we need to add an adapter for the object referenced by the provided **Reference**(p. 378).

- void **addChild** (const **Reference** &ref)

  Adds a child adapter (if necessary) to this adapter.

- void **removeChild** (const **Reference** &ref)

  Removes a child adapter from this adapter.

Private Attributes

- int64_t **mId**

  The id of the **Session**(p. 408) *this ContainerChangeTrackerAdapter belongs to. Used in order to keep track of which changes we should register and which changes that our own* **Session**(p. 408) *was originator to.*

- const **Reference** & **mReference**

  The **Reference**(p. 378) *to the adapted* **ContainerDataObject**(p. 129).

- bool **mChanged**

  Caches the result from the **changed()**(p. 127) *function until the next call to* **toXML()**(p. 128).

- std::map< const **Reference** *, char > **mChanges**

  When a child to the **DataObject**(p. 145) *this adapter adapts is added, removed or replaced, this is stored in the mChanges map. See* **objectAdded()**(p. 127) *for more information.*

- std::map< const **Reference** *, **ChangeTrackerAdapter** * > **mChildren**

  The child adapters.

4.31.1 Detailed Description

The ContainerChangeTrackerAdapter keeps track of changes in StratmasContainer objects.

ContainerChangeTrackerAdapter gets notified by the **SOFactory**(p. 436) when SimulationObjects are created, deleted or replaced.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.31.2 Constructor & Destructor Documentation

4.31.2.1 ContainerChangeTrackerAdapter::ContainerChangeTrackerAdapter (const Reference & *ref*, int64_t *id*)

Creates a ContainerChangeTrackerAdapter for the **DataObject**(p. 145) the provided **Reference**(p. 378) refers to..

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) to track changes for.
id The id of the **Session**(p. 408) this adapter belongs to.

4.31.3 Member Function Documentation

4.31.3.1 bool ContainerChangeTrackerAdapter::addable (const Reference & *ref*) [private]

Checks if we need to add an adapter for the object referenced by the provided **Reference**(p. 378).

For example - we don't have to create adapters for objects that won't change, such as Cities (Populations).

Parameters:

ref The **Reference**(p. 378) to the object to check.

Returns:

True if we need an adapter for the object refered to by ref, false otherwise.

4.31.3.2 void ContainerChangeTrackerAdapter::addChild (const Reference & *ref*) [private]

Adds a child adapter (if necessary) to this adapter.

Parameters:

ref The **Reference**(p. 378) to the object to add an adapter for.

4.31.3.3 bool ContainerChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 128) function.

The adapted **ContainerDataObject**(p. 129) is considered to be changed if any of its children has been modifed, added, removed or replaced.

Returns:

True if the **ContainerDataObject**(p. 129) this adapter adapts has changed since the last call to the **toXML()**(p. 128) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.31.3.4 void ContainerChangeTrackerAdapter::objectAdded (const Reference & *ref*, int64_t *initiator*) [virtual]

Notifies this change tracker that an object has been added.

A ContainerChangeTrackerAdapter only needs to store one entry per child (optionals and listelements) about what has happend to the object since the last call to **toXML()**(p. 128). The table below shows how an entry will change depending on current state and the event that occurred (0 = nothing, a = add, r = remove, x = exchange/replace).

Gets		a	r	x
Has	0	a	r	x
	a	-	0	a
	r	x	-	-
	x	-	r	r

Parameters:

ref The reference to the added object

initiator The initiator of the event.

Implements **SOFactoryListener** (p. 450).

4.31.3.5 void ContainerChangeTrackerAdapter::objectRemoved (const Reference & *ref*, int64_t *initiator*) [virtual]

Notifies this change tracker that an object has been removed. See **objectAdded()**(p. 127) for more information.

Parameters:

ref The reference to the removed object

initiator The initiator of the event.

Implements **SOFactoryListener** (p. 451).

4.31.3.6 void ContainerChangeTrackerAdapter::removeChild (const Reference & *ref*) [private]

Removes a child adapter from this adapter.

Parameters:

ref The **Reference**(p. 378) to the object to remove an adapter for.

4.31.3.7 ostream & ContainerChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

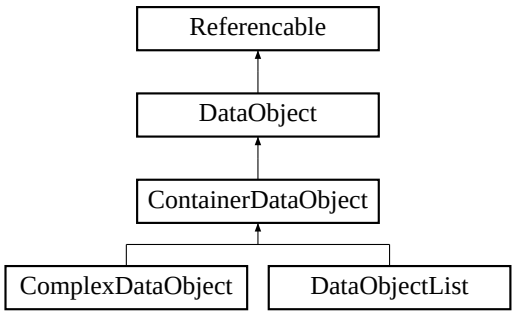
- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.32 ContainerDataObject Class Reference

ContainerDataObject is the abstract super class for all lists and complex objects in the Stratmas xml schema except ValueType descendants.

`#include <DataObjectImpl.h>`

Inheritance diagram for ContainerDataObject::



Public Member Functions

- const std::vector< **DataObject** * > & **objects** () const
  Returns the children of this **DataObject**(p. 145).
- bool **hasChildren** () const
  Checks if this **DataObject**(p. 145) *has children.*
- void **add** (**DataObject** *o)
  Adds a child to this ContainerDataObject.
- void **remove** (const std::string &name)
  Removes a child from this ContainerDataObject.
- void **replace** (**DataObject** *o)
  Replaces a child in this ContainerDataObject.
- **DataObject** * **getChild** (const std::string &name) const
  Returns the child with the specified identifier or null if there is no such child.
- void **reg** () const
  Registers this ContainerDataObject and all its children with the **Mapper**(p. 296).
- void **dereg** () const
  Deregisters this ContainerDataObject and all its children from the **Mapper**(p. 296).
- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const
  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*
- virtual void **print** (std::ostream &o, const std::string indent) const
  For debug purposes.

Protected Member Functions

- **ContainerDataObject** (const **ContainerDataObject** &c)

  Copy constructor.

- **ContainerDataObject** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **ContainerDataObject** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

Protected Attributes

- std::vector< **DataObject** * > **mObjects**

  A vector with the children of this container. The order of the children is the same as in the xml schema.

- std::map< std::string, **DataObject** * > **mObjectMap**

  Maps a child's identifier to the child itself.

4.32.1 Detailed Description

ContainerDataObject is the abstract super class for all lists and complex objects in the Stratmas xml schema except ValueType descendants.

Author:
Per Alexius

Date:
2006/03/27 09:43:40

4.32.2 Constructor & Destructor Documentation

4.32.2.1 ContainerDataObject::ContainerDataObject (const ContainerDataObject & *c*) [protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.32.2.2 ContainerDataObject::ContainerDataObject (const Reference & *scope*, const DOMElement * *n*) [inline, protected]

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:

scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.32.2.3 ContainerDataObject::ContainerDataObject (const Reference & *ref*, const Type & *type*) [inline, protected]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:

ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.32.3 Member Function Documentation

4.32.3.1 void ContainerDataObject::add (DataObject * *o*)

Adds a child to this ContainerDataObject.

Parameters:

o The child to add.

4.32.3.2 ostream & ContainerDataObject::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.32.3.3 DataObject * ContainerDataObject::getChild (const std::string & *name*) const [virtual]

Returns the child with the specified identifier or null if there is no such child.

Parameters:

name The identifier of the child to get.

Returns:

The child with the specified identifier or null if there is no such child.

Reimplemented from **DataObject** (p. 149).

4.32.3.4 bool ContainerDataObject::hasChildren () const [inline, virtual]

Checks if this **DataObject**(p. 145) has children.

Returns:
True if this **DataObject**(p. 145) has children, false otherwise.

Reimplemented from **DataObject** (p. 151).

4.32.3.5 const std::vector<DataObject*>& ContainerDataObject::objects () const [inline, virtual]

Returns the children of this **DataObject**(p. 145).

Returns:
The children of this **DataObject**(p. 145).

Reimplemented from **DataObject** (p. 151).

4.32.3.6 void ContainerDataObject::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:
o The ostream to print to.
indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.32.3.7 void ContainerDataObject::remove (const std::string & *name*)

Removes a child from this ContainerDataObject.

Parameters:
name The identifier of the child to remove.

4.32.3.8 void ContainerDataObject::replace (DataObject * *newObj*)

Replaces a child in this ContainerDataObject.

Parameters:
newObj The object to replace the old object with.

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

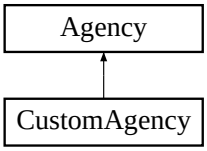
- DataObjectImpl.h
- DataObjectImpl.cpp

4.33 CustomAgency Class Reference

Class containing functionality for controlling CustomAgencyTeams.

`#include <Agency.h>`

Inheritance diagram for CustomAgency::



Public Member Functions

- **CustomAgency** (const std::vector< **AgencyTeam** * > &teams, **Grid** &g)

  Constructor.

- void **setTeamsGoals** ()

  Determine positions of the teams.

- void **act** (**Time** now)

  Agency(p. 25) *over all default behaviour.*

Private Member Functions

- bool **severeProblem** ()

  Determines if we have a severe problem. If we do - then the weights for the clustering algorithm is set.

4.33.1 Detailed Description

Class containing functionality for controlling CustomAgencyTeams.

Author:
Per Alexius

Date:
2006/10/02 16:01:25

4.33.2 Constructor & Destructor Documentation

4.33.2.1 CustomAgency::CustomAgency (const std::vector< AgencyTeam * > & *teams*, Grid & *g*) [inline]

Constructor.

Parameters:

teams A vector containing this Agency's teams.

g A reference to the **Grid**(p. 227).

4.33.3 Member Function Documentation

4.33.3.1 void CustomAgency::act (Time *now*) [virtual]

Agency(p. 25) over all default behaviour.

The default behaviour is as follows:

For all teams that are not operational - set their start time.

For operational teams - Check if current time is later than their start time and if it is - consider them active teams.

For all active teams - calculate their need, divide the resources evenly among all teams and let each team supply their resources.

Parameters:

now The current simulation time.

Reimplemented from **Agency** (p. 27).

4.33.3.2 void CustomAgency::setTeamsGoals () [virtual]

Determine positions of the teams.

Assigns one team to each cluster found. If there are more teams than clusters - assign more than one team to each cluster.

Reimplemented from **Agency** (p. 28).

4.33.3.3 bool CustomAgency::severeProblem () [private, virtual]

Determines if we have a severe problem. If we do - then the weights for the clustering algorithm is set.

Returns:

True if we have a severe violence problem, false otherwise.

Implements **Agency** (p. 28).

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

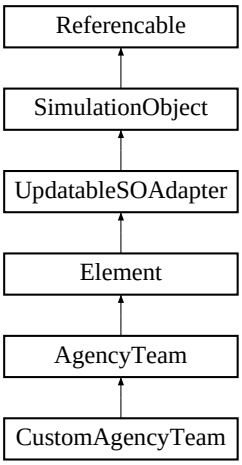
- Agency.h
- Agency.cpp

4.34 CustomAgencyTeam Class Reference

Class representing a CustomAgencyTeam.

`#include <AgencyTeam.h>`

Inheritance diagram for CustomAgencyTeam::



Public Member Functions

- **CustomAgencyTeam** (const **DataObject** &d)

  Constructor that creates an **AgencyTeam**(p. 32) *from the provided* **DataObject**(p. 145).

- void **addEffect** (eAllPV a, double si, **EthnicFaction** *f)

  Adds an effect to this activity.

- virtual void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- virtual void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- virtual void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- virtual void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- virtual void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- void **act** (**Time** now)

  Performs this teams actions.

Private Member Functions

- void **createEffects** ()
  Creates the effects of this **AgencyTeam**(p. 32) *from the mSeverities map.*

Private Attributes

- const **Reference** * **mTarget**
  Reference(p. 378) *to the target* **Faction**(p. 212).
- std::vector< **GridEffect** > **mEffects**
  Vector of effects.
- std::map< std::string, double > **mSeverities**
  Map(p. 292) *that maps the PV name to the severity of the impact.*

4.34.1 Detailed Description

Class representing a CustomAgencyTeam.

A CustomAgenyTeam may be set to affect any combination of modifiable process variables in the area it occupies. The functionality is very similar to the **CustomPVModification**(p. 139).

This type of team was not present in old Stratmas. It was introduced to meet requirements that arose before the Demo06 exercise during the autumn 2006.

Author:
Per Alexius

Date **Date**
2006/10/10 09:35:59

4.34.2 Constructor & Destructor Documentation

4.34.2.1 CustomAgencyTeam::CustomAgencyTeam (const DataObject & *d*)

Constructor that creates an **AgencyTeam**(p. 32) from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.34.3 Member Function Documentation

4.34.3.1 void CustomAgencyTeam::act (Time *now*) [virtual]

Performs this teams actions.

Parameters:
now The current simulation time.

Implements **AgencyTeam** (p. 35).

4.34.3.2 void CustomAgencyTeam::addEffect (eAllPV *a*, double *si*, EthnicFaction * *f*) [inline]

Adds an effect to this activity.

Parameters:

a The effect to add.

si The magnitude of the effect to add.

f The affected faction.

4.34.3.3 void CustomAgencyTeam::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:

toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.

initiator The id of the initiator of the update.

Reimplemented from **AgencyTeam** (p. 36).

4.34.3.4 void CustomAgencyTeam::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:

b The **Buffer**(p. 67) to extract data to.

Reimplemented from **AgencyTeam** (p. 37).

4.34.3.5 void CustomAgencyTeam::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) containing the new value.

Reimplemented from **AgencyTeam** (p. 37).

4.34.3.6 void CustomAgencyTeam::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:

toRemove The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the update.

Reimplemented from **AgencyTeam** (p. 38).

4.34.3.7 void CustomAgencyTeam::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **AgencyTeam** (p. 38).

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

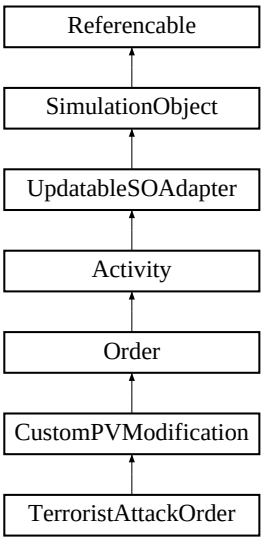
- AgencyTeam.h
- AgencyTeam.cpp

4.35 CustomPVModification Class Reference

The CustomPVModification activity.

`#include <Activity.h>`

Inheritance diagram for CustomPVModification::



Public Member Functions

- **CustomPVModification** (const **DataObject** &d)

  Creates a CustomPVModification object from the provided **DataObject**(p. 145).

- void **prepareForSimulation** (**Grid** &g, **Time** currentTime)

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- void **createEffects** ()

  Creates the effects of this **Activity**(p. 21) *from the mSeverities map.*

- virtual void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- virtual void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- virtual void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- virtual void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- virtual void **reset** (const **DataObject** &d)

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- **Time endTime** () const

  Accessor for the end time.

- virtual bool **isActive** (**Time** t)

  Checks if this activity is active at time t.

- void **addEffect** (eAllPV a, double si, **EthnicFaction** *f)

  Adds an effect to this activity.

- virtual void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

- virtual double **combatFactor** () const

  Accessor for the combat factor.

Protected Attributes

- **Grid** * **mGrid**

  The grid.

- **Time mEnd**

  The end time.

- const **Reference** * **mTarget**

  Reference(p. 378) *to the target* **Faction**(p. 212).

- std::vector< **GridEffect** > **mEffects**

  Vector of effects.

- std::string **mType**

  String representation of the type of this **Activity**(p. 21).

- double **mCombatFactor**

  The combat factor of this **Activity**(p. 21).

- std::map< std::string, double > **mSeverities**

  Map(p. 292) *that maps the PV name to the severity of the impact.*

Friends

- std::ostream & **operator**<< (std::ostream &o, const **CustomPVModification** &a)

  For debugging purposes.

4.35.1 Detailed Description

The CustomPVModification activity.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.35.2 Constructor & Destructor Documentation

4.35.2.1 CustomPVModification::CustomPVModification (const DataObject & *d*)

Creates a CustomPVModification object from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.35.3 Member Function Documentation

4.35.3.1 void CustomPVModification::addEffect (eAllPV *a*, double *si*, EthnicFaction * *f*) [inline]

Adds an effect to this activity.

Parameters:
a The effect to add.
si The magnitude of the effect to add.
f The affected faction.

4.35.3.2 void CustomPVModification::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:
toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.
initiator The id of the initiator of the update.

Reimplemented from **Order** (p. 309).

4.35.3.3 virtual double CustomPVModification::combatFactor () const [inline, virtual]

Accessor for the combat factor.

Returns:
The combat factor.

Implements **Order** (p. 310).

4.35.3.4 Time CustomPVModification::endTime () const [inline]

Accessor for the end time.

Returns:
The emd time.

4.35.3.5 void CustomPVModification::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Reimplemented from **Order** (p. 310).

Reimplemented in **TerroristAttackOrder** (p. 514).

4.35.3.6 virtual bool CustomPVModification::isActive (Time *t*) [inline, virtual]

Checks if this activity is active at time t.

Parameters:
t The time for which to check.

Returns:
True if this activity is active at the specified time.

Reimplemented from **Order** (p. 310).

Reimplemented in **TerroristAttackOrder** (p. 515).

4.35.3.7 void CustomPVModification::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **Order** (p. 311).

Reimplemented in **TerroristAttackOrder** (p. 515).

4.35.3.8 void CustomPVModification::perform (Element * *e*, double *fraction* = 1.0) [virtual]

Performs this **Activity**(p. 21).

Parameters:
e The **Element**(p. 180) that should perform this **Activity**(p. 21).
fraction The fraction of the performers total capacity that this activity is performed with.

Implements **Activity** (p. 23).

Reimplemented in **TerroristAttackOrder** (p. 515).

4.35.3.9 void CustomPVModification::prepareForSimulation (Grid & *g*, Time *currentTime*) [virtual]

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Parameters:

g The **Grid**(p. 227).

currentTime The current simulation time.

Reimplemented from **Activity** (p. 24).

4.35.3.10 void CustomPVModification::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:

toRemove The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the update.

Reimplemented from **Order** (p. 311).

4.35.3.11 void CustomPVModification::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Order** (p. 312).

Reimplemented in **TerroristAttackOrder** (p. 515).

4.35.4 Friends And Related Function Documentation

4.35.4.1 std::ostream& operator<< (std::ostream & *o*, const CustomPVModification & *a*) [friend]

For debugging purposes.

Parameters:

o The stream to write to.

a The activity to print.

Returns:

The provided ostream with the **Activity**(p. 21) written to it.

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

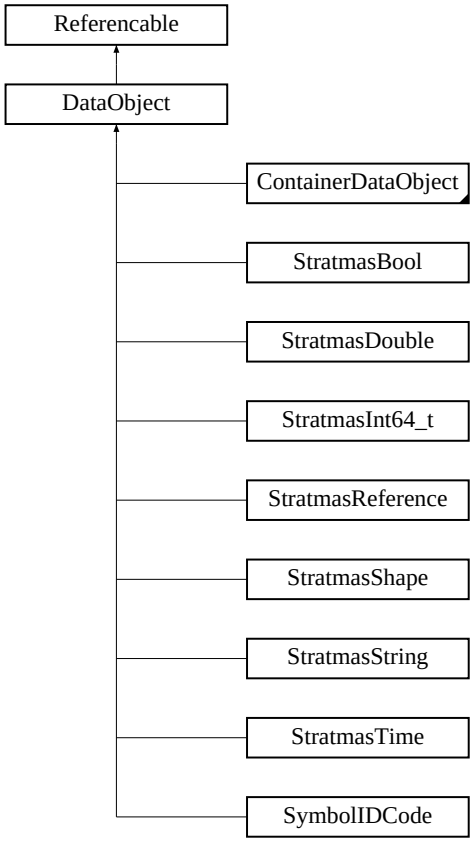
- Activity.h
- Activity.cpp

4.36 DataObject Class Reference

This is the abstract super class for all types of DataObjects. A DataObject is the kind of object that is used to store the data that is communicated with the client.

`#include <DataObject.h>`

Inheritance diagram for DataObject::



Public Member Functions

- virtual ~**DataObject** ()
  Destructor.
- const **Type** & **getType** () const
  Accessor for the **Type**(p. 528).
- const std::string & **identifier** () const
  Accessor for the identifier.
- virtual const std::vector< **DataObject** * > & **objects** () const
  Returns the children of this DataObject.
- virtual bool **hasChildren** () const
  Checks if this DataObject has children.

- virtual **DataObject** * **getChild** (const std::string &name) const

  Returns the child with the specified identifier or null if there is no such child.

- void **setParent** (**ContainerDataObject** *parent)

  Sets the parent of this DataObject.

- **ContainerDataObject** * **getParent** () const

  Accessor for the parent.

- virtual void **setBool** (bool v)

  Mutator.

- virtual void **setDouble** (double v)

  Mutator.

- virtual void **setInt64_t** (int64_t v)

  Mutator.

- virtual void **setTime** (**Time** v)

  Mutator.

- virtual void **setString** (const std::string &v)

  Mutator.

- virtual void **setReference** (const **Reference** &v)

  Mutator.

- virtual void **setShape** (const **Shape** *v)

  Mutator.

- virtual bool **getBool** () const

  Accessor.

- virtual double **getDouble** () const

  Accessor.

- virtual int64_t **getInt64_t** () const

  Accessor.

- virtual **Time** **getTime** () const

  Accessor.

- virtual std::string **getString** () const

  Accessor.

- virtual const **Reference** & **getReference** () const

  Accessor.

- virtual **Shape** * **getShape** () const

Accessor.

- virtual **DataObject** & **operator=** (const **DataObject** &d)
  Assignment operator.
- void **print** (std::ostream &o) const
  For debugging purpooses.
- virtual void **print** (std::ostream &o, const std::string indent) const
  For debug purposes.
- virtual **DataObject** * **clone** () const =0
  Creates a clone of this DataObject.
- virtual void **reg** () const
  Registers this DataObject with the **Mapper**(p. 296).
- virtual void **dereg** () const
  Deregisters this DataObject from the **Mapper**(p. 296).
- virtual std::ostream & **toXML** (std::ostream &o) const
  Produces an XML representation of this DataObject according to the xml schemas.
- virtual std::ostream & **toXML** (std::ostream &o, std::string indent) const
  Produces an XML representation of this DataObject according to the Stratmas xml schemas.
- virtual std::ostream & **bodyXML** (std::ostream &o, std::string indent) const =0
  Produces an XML representation of the body of this DataObject according to the xml schemas.

Protected Member Functions

- **DataObject** (const **Reference** &scope, const DOMElement *n)
  Constructor that creates a DataObject in the provided scope from the provided DOMElement.
- **DataObject** (const **Reference** &ref, const **Type** &type)
  Constructor that creates a DataObject of the specified **Type**(p. 528) *with the provided* **Reference**(p. 378).
- **DataObject** (const **DataObject** &c)
  Copy constuctor.

Private Attributes

- const **Type** & **mType**
  The **Type**(p. 528) *of the DataObject.*
- **ContainerDataObject** * **mParent**
  The parent of this DataObject.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **DataObject** &d)

  For debugging purpooses.

4.36.1 Detailed Description

This is the abstract super class for all types of DataObjects. A DataObject is the kind of object that is used to store the data that is communicated with the client.

DataObjects are created according to the Stratmas xml schema. They are then used to create the corresponding SimulationObjects

Author:
Per Alexius

Date **Date**
2006/05/24 12:32:10

4.36.2 Constructor & Destructor Documentation

4.36.2.1 DataObject::DataObject (const Reference & *scope*, const DOMElement * *n*) [protected]

Constructor that creates a DataObject in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the DataObject in.

n The DOMElement to create this DataObject from.

4.36.2.2 DataObject::DataObject (const Reference & *ref*, const Type & *type*) [inline, protected]

Constructor that creates a DataObject of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the DataObject tp be created.

type The **Type**(p. 528) of the DataObject to be created.

4.36.2.3 DataObject::DataObject (const DataObject & *c*) [inline, protected]

Copy constuctor.

Parameters:
c The DataObject to copy.

4.36.3 Member Function Documentation

4.36.3.1 virtual std::ostream& DataObject::bodyXML (std::ostream & *o*, std::string *indent*) const [pure virtual]

Produces an XML representation of the body of this DataObject according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implemented in **ContainerDataObject** (p. 131), **StratmasBool** (p. 462), **StratmasDouble** (p. 466), **StratmasInt64_t** (p. 470), **StratmasReference** (p. 478), **StratmasShape** (p. 483), **StratmasString** (p. 490), **StratmasTime** (p. 494), and **SymbolIDCode** (p. 504).

4.36.3.2 virtual DataObject* DataObject::clone () const [pure virtual]

Creates a clone of this DataObject.

Returns:

A clone of this DataObject.

Implemented in **ComplexDataObject** (p. 111), **DataObjectList** (p. 158), **StratmasBool** (p. 462), **StratmasDouble** (p. 466), **StratmasInt64_t** (p. 470), **StratmasReference** (p. 478), **StratmasShape** (p. 483), **StratmasString** (p. 490), **StratmasTime** (p. 494), and **SymbolIDCode** (p. 504).

4.36.3.3 bool DataObject::getBool () const [virtual]

Accessor.

Returns:

The current value.

Reimplemented in **StratmasBool** (p. 462).

4.36.3.4 virtual DataObject* DataObject::getChild (const std::string & *name*) const [inline, virtual]

Returns the child with the specified identifier or null if there is no such child.

Parameters:

name The identifier of the child to get.

Returns:

The child with the specified identifier or null if there is no such child.

Reimplemented in **ContainerDataObject** (p. 131).

4.36.3.5 double DataObject::getDouble () const [virtual]

Accessor.

Returns:
The current value.

Reimplemented in **StratmasDouble** (p. 466).

4.36.3.6 int64_t DataObject::getInt64_t () const [virtual]

Accessor.

Returns:
The current value.

Reimplemented in **StratmasInt64_t** (p. 470).

4.36.3.7 ContainerDataObject* DataObject::getParent () const [inline]

Accessor for the parent.

Returns:
The parent.

4.36.3.8 const Reference & DataObject::getReference () const [virtual]

Accessor.

Returns:
The current value.

Reimplemented in **StratmasReference** (p. 478).

4.36.3.9 Shape * DataObject::getShape () const [virtual]

Accessor.

Returns:
The current value.

Reimplemented in **StratmasShape** (p. 483).

4.36.3.10 string DataObject::getString () const [virtual]

Accessor.

Returns:
The current value.

Reimplemented in **StratmasString** (p. 490), and **SymbolIDCode** (p. 504).

4.36.3.11 Time DataObject::getTime () const [virtual]

Accessor.

Returns:
The current value.

Reimplemented in **StratmasTime** (p. 494).

4.36.3.12 const Type& DataObject::getType () const [inline]

Accessor for the **Type**(p. 528).

Returns:
The **Type**(p. 528) of this DataObject.

4.36.3.13 virtual bool DataObject::hasChildren () const [inline, virtual]

Checks if this DataObject has children.

Returns:
True if this DataObject has children, false otherwise.

Reimplemented in **ContainerDataObject** (p. 132).

4.36.3.14 const std::string & DataObject::identifier () const

Accessor for the identifier.

Returns:
The identifier of this DataObject.

4.36.3.15 const vector< DataObject * > & DataObject::objects () const [virtual]

Returns the children of this DataObject.

Returns:
The children of this DataObject.

Reimplemented in **ContainerDataObject** (p. 132).

4.36.3.16 DataObject & DataObject::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:
d The object to copy.

Returns:
The assigned object.

Reimplemented in **StratmasBool** (p. 462), **StratmasDouble** (p. 466), **StratmasInt64_t** (p. 470), **StratmasReference** (p. 478), **StratmasShape** (p. 483), **StratmasString** (p. 490), **StratmasTime** (p. 494), and **SymbolIDCode** (p. 504).

4.36.3.17 void DataObject::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:
o The ostream to print to.
indent Intention string for readable output.

Reimplemented in **ContainerDataObject** (p. 132), **StratmasBool** (p. 463), **StratmasDouble** (p. 467), **StratmasInt64_t** (p. 471), **StratmasReference** (p. 479), **StratmasShape** (p. 484), **StratmasString** (p. 491), **StratmasTime** (p. 495), and **SymbolIDCode** (p. 505).

4.36.3.18 void DataObject::print (std::ostream & *o*) const [inline]

For debugging purpooses.

Parameters:
o The ostream to write to.

4.36.3.19 void DataObject::setBool (bool *v*) [virtual]

Mutator.

Parameters:
v The new value.

Reimplemented in **StratmasBool** (p. 463).

4.36.3.20 void DataObject::setDouble (double *v*) [virtual]

Mutator.

Parameters:
v The new value.

Reimplemented in **StratmasDouble** (p. 467).

4.36.3.21 void DataObject::setInt64_t (int64_t *v*) [virtual]

Mutator.

Parameters:
v The new value.

Reimplemented in **StratmasInt64_t** (p. 471).

4.36.3.22 void DataObject::setParent (ContainerDataObject * *parent*) [inline]

Sets the parent of this DataObject.

Parameters:
parent The new parent.

4.36.3.23 void DataObject::setReference (const Reference & *v*) [virtual]

Mutator.

Parameters:
v The new value.

Reimplemented in **StratmasReference** (p. 479).

4.36.3.24 void DataObject::setShape (const Shape * *v*) [virtual]

Mutator.

Parameters:
v The new value.

Reimplemented in **StratmasShape** (p. 484).

4.36.3.25 void DataObject::setString (const std::string & *v*) [virtual]

Mutator.

Parameters:
v The new value.

Reimplemented in **StratmasString** (p. 491), and **SymbolIDCode** (p. 505).

4.36.3.26 void DataObject::setTime (Time *v*) [virtual]

Mutator.

Parameters:
v The new value.

Reimplemented in **StratmasTime** (p. 495).

4.36.3.27 ostream & DataObject::toXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of this DataObject according to the Stratmas xml schemas.

Parameters:
o The ostream to print to.

indent Intention string for readable output.

Returns:
The ostream with the XML representation written to it.

Reimplemented in **DataObjectList** (p. 159).

4.36.3.28 virtual std::ostream& DataObject::toXML (std::ostream & *o*) const [inline, virtual]

Produces an XML representation of this DataObject according to the xml schemas.

Parameters:
o The ostream to print to.

Returns:
The ostream with the XML representation written to it.

4.36.4 Friends And Related Function Documentation

4.36.4.1 std::ostream& operator<< (std::ostream & *o*, const DataObject & *d*) [friend]

For debugging purpooses.

Parameters:
o The ostream to write to.
d The DataObject to write.

Returns:
The ostream with the DataObject written to it.

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

- DataObject.h
- DataObjectImpl.cpp

4.37 DataObjectFactory Class Reference

Factory for creating DataObjects.

`#include <DataObject.h>`

Static Public Member Functions

- **DataObject** * **createDataObject** (const **Reference** &scope, const DOMElement *n)
  Creates a **DataObject**(p. 145) *from the provided DOMElement.*
- **DataObject** * **createDataObject** (const **Reference** &ref, const **Type** &type)
  Creates a default **DataObject**(p. 145).
- **DataObject** * **addOptional** (**DataObject** &parent, const std::string &idToAdd)
  Adds an optional element to the provided **DataObject**(p. 145).
- void **addObjectTo** (const **Reference** &parent, **DataObject** &objToAdd)
  Convenience function for adding a **DataObject**(p. 145) *to the* **DataObject**(p. 145) *referenced by the provided* **Reference**(p. 378).

4.37.1 Detailed Description

Factory for creating DataObjects.

Author:
Per Alexius

Date:
2006/05/24 12:32:10

4.37.2 Member Function Documentation

4.37.2.1 void DataObjectFactory::addObjectTo (const Reference & *parent*, DataObject & *objToAdd*) [static]

Convenience function for adding a **DataObject**(p. 145) to the **DataObject**(p. 145) referenced by the provided **Reference**(p. 378).

Parameters:
parent **Reference**(p. 378) to the parent to add to.
objToAdd The **DataObject**(p. 145) to add.

4.37.2.2 DataObject * DataObjectFactory::addOptional (DataObject & *parent*, const std::string & *idToAdd*) [static]

Adds an optional element to the provided **DataObject**(p. 145).

Parameters:

parent The **DataObject**(p. 145) to add an optional element to.

idToAdd The name of the optional element to add.

4.37.2.3 DataObject * DataObjectFactory::createDataObject (const Reference & *ref*, const Type & *type*) [static]

Creates a default **DataObject**(p. 145).

Parameters:

ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

Returns:

The newly created **DataObject**(p. 145).

4.37.2.4 DataObject * DataObjectFactory::createDataObject (const Reference & *scope*, const DOMElement * *n*) [static]

Creates a **DataObject**(p. 145) from the provided DOMElement.

Parameters:

n The DOMElement to create the **DataObject**(p. 145) from.

scope The **Reference**(p. 378) to the scope this **DataObject**(p. 145) should live in.

Returns:

The newly created **DataObject**(p. 145).

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

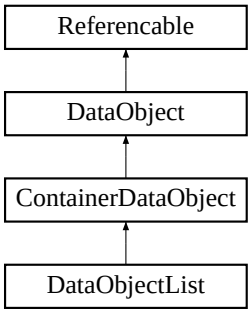
- DataObject.h
- DataObjectImpl.cpp

4.38 DataObjectList Class Reference

DataObjectsList represent lists in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for DataObjectList::



Public Member Functions

- **DataObjectList** (const **Reference** &scope, const **Declaration** &dec, const DOMElement *n)

  Constructor that creates a DataObjectList in the provided scope from the provided DOMElement based on the provided **Declaration**(p. 160).

- **DataObjectList** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **toXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of this **DataObject**(p. 145) *according to the Stratmas xml schemas.*

Protected Member Functions

- **DataObjectList** (const **DataObjectList** &c)

  Copy constructor.

4.38.1 Detailed Description

DataObjectsList represent lists in the Stratmas xml schema.

Author:
Per Alexius

Date Date

2006/03/27 09:43:40

4.38.2 Constructor & Destructor Documentation

4.38.2.1 DataObjectList::DataObjectList (const DataObjectList & *c*) [inline, protected]

Copy constructor.

Parameters:

c The **DataObject**(p. 145) to copy.

4.38.2.2 DataObjectList::DataObjectList (const Reference & *scope*, const Declaration & *dec*, const DOMElement * *n*)

Constructor that creates a DataObjectList in the provided scope from the provided DOMElement based on the provided **Declaration**(p. 160).

Parameters:

scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

dec The **Declaration**(p. 160) to use when creating this list.

n The DOMElement to create this **DataObject**(p. 145) from.

4.38.2.3 DataObjectList::DataObjectList (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:

ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.38.3 Member Function Documentation

4.38.3.1 DataObject* DataObjectList::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.38.3.2 ostream & DataObjectList::toXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of this **DataObject**(p. 145) according to the Stratmas xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Reimplemented from **DataObject** (p. 153).

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

- DataObjectImpl.h
- DataObjectImpl.cpp

4.39 Declaration Class Reference

This class represents an element declaration in the xml schema.

`#include <Declaration.h>`

Public Member Functions

- **Declaration** (XSParticle &particle, **XSDContent** &xsdcontent)

  Creates a Declaration.

- **Declaration** (XSParticle &particle, const **Type** *type)

  Creates a Declaration of the specified **Type**(p. 528). *eccesary in order to avoid infinite recursion for recursively defined types.*

- int **minOccurs** () const

  Accessor for the minOccurs attribute.

- int **maxOccurs** () const

  Accessor for the axOccurs attribute.

- int **unbounded** () const

  Accessor for the unbounded flag.

- bool **isList** () const

  Checks if this Declaration refers to a list i.e. if minOccurs and macOccurs differ more than 1.

- bool **isOptional** () const

  Checks if this Declaration refers to an optional element, i.e. if minOccurs = 0 and maxOccurs = 1.

- bool **isSingular** () const

  Checks if this Declaration refers to a singular element, i.e. if minOccurs = 1 and maxOccurs = 1.

- const std::string & **getName** () const

  Accessor for the name.

- const **Type** & **getType** () const

  Accessor for the **Type**(p. 528).

Private Member Functions

- void **init** (XSParticle &particle, **XSDContent** *xsdcontent, const **Type** *type)

  Helper function for initializing the Declaration.

Private Attributes

- int **mMinOccurs**

  The minOccurs attribute.

- int **mMaxOccurs**

  The maxOccurs attribute.

- bool **mUnbounded**

  Flag indicating unbounded or not.

- std::string **mName**

  The name of the Declaration.

- const **Type** * **mType**

  The **Type**(p. 528) *of the Declaration.*

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Declaration** &d)

  For debugging purposes.

4.39.1 Detailed Description

This class represents an element declaration in the xml schema.

Author:
Per Alexius

Date **Date**
2006/05/24 12:32:10

4.39.2 Constructor & Destructor Documentation

4.39.2.1 Declaration::Declaration (XSParticle & *particle*, XSDContent & *xsdcontent*)

Creates a Declaration.

Parameters:
particle The XSParticle to create the Declaration from.

xsdcontent The **XSDContent**(p. 600) to create the Declaration from.

4.39.2.2 Declaration::Declaration (XSParticle & *particle*, const Type * *type*)

Creates a Declaration of the specified **Type**(p. 528). eccesary in order to avoid infinite recursion for recursively defined types.

Parameters:

particle The XSParticle to create the Declaration from.

type The **Type**(p. 528) of the Declaration.

4.39.3 Member Function Documentation

4.39.3.1 const std::string& Declaration::getName () const [inline]

Accessor for the name.

Returns:

The name of this Declaration.

4.39.3.2 const Type& Declaration::getType () const [inline]

Accessor for the **Type**(p. 528).

Returns:

The **Type**(p. 528) of this Declaration.

4.39.3.3 void Declaration::init (XSParticle & *particle*, XSDContent * *xsdcontent*, const Type * *type*) [private]

Helper function for initializing the Declaration.

Parameters:

particle The XSParticle to create the Declaration from.

xsdcontent The **XSDContent**(p. 600) to create the Declaration from.

type The **Type**(p. 528) of the Declaration.

4.39.3.4 bool Declaration::isList () const [inline]

Checks if this Declaration refers to a list i.e. if minOccurs and macOccurs differ more than 1.

Returns:

True if this Delcaration refers to a list, false otherwise.

4.39.3.5 bool Declaration::isOptional () const [inline]

Checks if this Declaration refers to an optional element, i.e. if minOccurs = 0 and maxOccurs = 1.

Returns:

True if this Delcaration refers to an optional element, false otherwise.

4.39.3.6 bool Declaration::isSingular () const [inline]

Checks if this Declaration refers to a singular element, i.e. if minOccurs = 1 and maxOccurs = 1.

Returns:
True if this Delcaration refers to a singular element, false otherwise.

4.39.3.7 int Declaration::maxOccurs () const [inline]

Accessor for the axOccurs attribute.

Returns:
The maxOccurs attribute.

4.39.3.8 int Declaration::minOccurs () const [inline]

Accessor for the minOccurs attribute.

Returns:
The minOccurs attribute.

4.39.3.9 int Declaration::unbounded () const [inline]

Accessor for the unbounded flag.

Returns:
The state of the unbounded flag.

4.39.4 Friends And Related Function Documentation

4.39.4.1 std::ostream& operator<< (std::ostream & *o*, const Declaration & *d*) [friend]

For debugging purposes.

Parameters:
o The stream to write to.
d The Declaration to print.

Returns:
The provided ostream with the Decaration written to it.

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

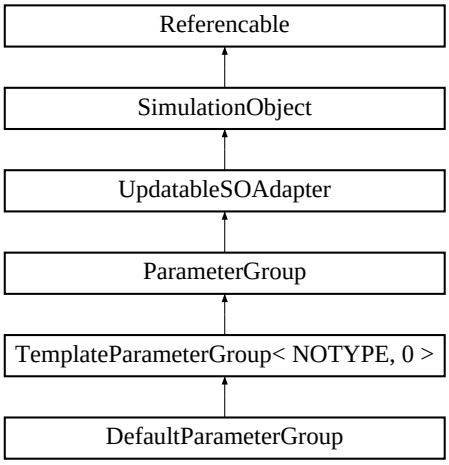
- Declaration.h
- Declaration.cpp

4.40 DefaultParameterGroup Class Reference

The default parameter group for the simulation.

`#include <ValidParameterGroups.h>`

Inheritance diagram for DefaultParameterGroup::



Public Member Functions

- **DefaultParameterGroup** (const **DataObject** &d)

4.40.1 Detailed Description

The default parameter group for the simulation.

Creation of this object will also create the necessary child parameter groups that are not already created.

Author:
Per Alexius

Date:
2007/01/24 13:13:25

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

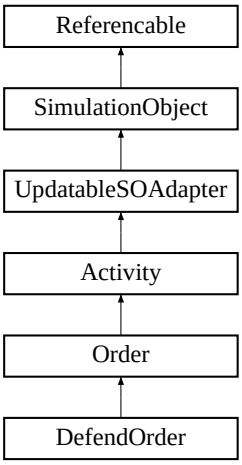
- ValidParameterGroups.h

4.41 DefendOrder Class Reference

The DefendOrder.

`#include <Activity.h>`

Inheritance diagram for DefendOrder::



Public Member Functions

- **DefendOrder** (const **DataObject** &d)

  Creates a DefendOrder object from the provided **DataObject**(p. 145).

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- bool **isActive** (**Time** t)

  Checks if this activity is active at time t.

- void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

- double **combatFactor** () const

  Accessor for the combat factor.

Protected Attributes

- **Time mEnd**

The end time.

4.41.1 Detailed Description

The DefendOrder.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.41.2 Constructor & Destructor Documentation

4.41.2.1 DefendOrder::DefendOrder (const DataObject & *d*)

Creates a DefendOrder object from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) from which to create this object.

4.41.3 Member Function Documentation

4.41.3.1 double DefendOrder::combatFactor () const [inline, virtual]

Accessor for the combat factor.

Returns:
The combat factor.

Implements **Order** (p. 310).

4.41.3.2 void DefendOrder::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Reimplemented from **Order** (p. 310).

4.41.3.3 bool DefendOrder::isActive (Time *t*) [inline, virtual]

Checks if this activity is active at time t.

Parameters:
t The time for which to check.

Returns:
True if this activity is active at the specified time.

Reimplemented from **Order** (p. 310).

4.41.3.4 void DefendOrder::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) containing the new value.

Reimplemented from **Order** (p. 311).

4.41.3.5 void DefendOrder::perform (Element * *e*, double *fraction* = 1.0) [virtual]

Performs this **Activity**(p. 21).

Parameters:

e The **Element**(p. 180) that should perform this **Activity**(p. 21).

fraction The fraction of the performers total capacity that this activity is performed with.

Implements **Activity** (p. 23).

4.41.3.6 void DefendOrder::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Order** (p. 312).

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

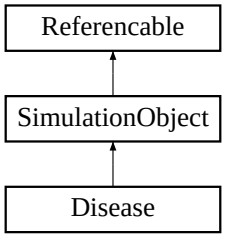
- Activity.h
- Activity.cpp

4.42 Disease Class Reference

This is the **SimulationObject**(p. 429) that corresponds to the Disease type in the Stratmas xml schema.

`#include <Disease.h>`

Inheritance diagram for Disease::



Public Member Functions

- **Disease** (const **DataObject** &d)

  Creates a Disease from the provided **DataObject**(p. 145).

- double **infectionRate** () const

  Accessor for the infection rate.

- double **recoveryRate** () const

  Accessor for the recovery rate.

- double **mortalityRate** () const

  Accessor for the mortality rate.

- void **update** (const **Update** &u)

  Updates this object.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Private Attributes

- std::string **mDescription**

  Description of the disease.

- double **mInfectionRate**

  The infection rate of this disease.

- double **mRecoveryRate**

  The recovery rate parameter of this disease.

- double **mMortalityRate**

  The mortality rate of this disease.

4.42.1 Detailed Description

This is the **SimulationObject**(p. 429) that corresponds to the Disease type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/03/06 09:18:08

4.42.2 Constructor & Destructor Documentation

4.42.2.1 Disease::Disease (const DataObject & *d*)

Creates a Disease from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.42.3 Member Function Documentation

4.42.3.1 void Disease::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.42.3.2 double Disease::infectionRate () const [inline]

Accessor for the infection rate.

Returns:
The infection rate.

4.42.3.3 double Disease::mortalityRate () const [inline]

Accessor for the mortality rate.

Returns:
The mortality rate.

4.42.3.4 double Disease::recoveryRate () const [inline]

Accessor for the recovery rate.

Returns:
The recovery rate.

4.42.3.5 void Disease::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

4.42.3.6 void Disease::update (const Update & *u*) [virtual]

Updates this object.

Parameters:
u The **Update**(p. 567) to update this object with.

Implements **SimulationObject** (p. 431).

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

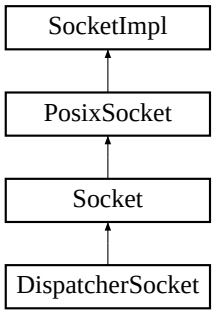
- Disease.h
- Disease.cpp

4.43 DispatcherSocket Class Reference

This class represents a connection to a dispatcher.

`#include <Registrator.h>`

Inheritance diagram for DispatcherSocket::



Public Member Functions

- **DispatcherSocket** (std::string host, int port)

  Creates a socket that connects to the specified host and port.

- bool **sendDispatcherMessage** (const std::string msg)

  Sends a stratmas message.

4.43.1 Detailed Description

This class represents a connection to a dispatcher.

Author:
Daniel Ahlin

Date:
2006/07/21 13:35:29

4.43.2 Constructor & Destructor Documentation

4.43.2.1 DispatcherSocket::DispatcherSocket (std::string *host*, int *port*)

Creates a socket that connects to the specified host and port.

Parameters:
host The host to connect to.

port The port to connect to.

4.43.3 Member Function Documentation

4.43.3.1 bool DispatcherSocket::sendDispatcherMessage (const std::string *msg*)

Sends a stratmas message.

Parameters:
msg The message to send.

Returns:
True if all is ok.

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

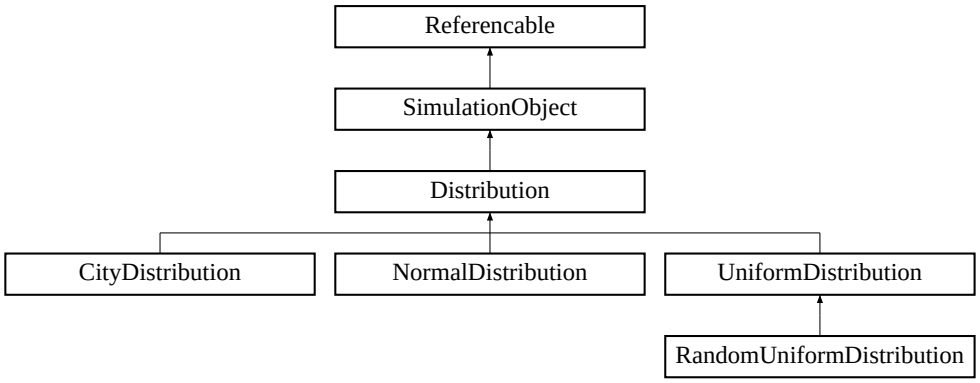
- Registrator.h
- Registrator.cpp

4.44 Distribution Class Reference

Abstract super class for all distributions.

`#include <Distribution.h>`

Inheritance diagram for Distribution::



Public Member Functions

- **Distribution** ()

  Default constructor.

- **Distribution** (const **DataObject** &d)

  Creates a Distribution from the provided **DataObject**(p. 145).

- virtual ~**Distribution** ()

  Destructor.

- virtual double **f** (double x) const =0

  Gets the value of the distribution at distance x.

- void **amount** (**LatLng** center, const std::list< **GridPos** > &cells, const **BasicGrid** &g, std::vector< double > &outAmount) const

  Calculates the fraction (of some entity) that goes to each of the cells in the provided list based on the distribution function and the distance between each cell and the specified center point.

- void **amountMean1** (**LatLng** center, const std::list< **GridPos** > &cells, const **BasicGrid** &g, std::vector< double > &outAmount) const

  Based on the distribution function and the distance between each cell and the specified center point this function fills the outAmount list with values so that the mean value is 1.

- virtual void **update** (const **Update** &u)

  Updates this object.

- virtual void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

4.44.1 Detailed Description

Abstract super class for all distributions.

Author:
Per Alexius

Date Date
2006/04/21 15:54:49

4.44.2 Constructor & Destructor Documentation

4.44.2.1 Distribution::Distribution (const DataObject & *d*) [inline]

Creates a Distribution from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.44.3 Member Function Documentation

4.44.3.1 void Distribution::amount (LatLng *center*, const std::list< GridPos > & *cells*, const BasicGrid & *g*, std::vector< double > & *outAmount*) const

Calculates the fraction (of some entity) that goes to each of the cells in the provided list based on the distribution function and the distance between each cell and the specified center point.

Parameters:
center The coorinate to center the Distribution around.

cells A list of cell positions for which to calculate the fraction.

g The grid that the cells belongs to.

outAmount On return this vector has the same size as the cells list and outAmount[i] contains the fraction for the cell specified by cells[i].

4.44.3.2 void Distribution::amountMean1 (LatLng *center*, const std::list< GridPos > & *cells*, const BasicGrid & *g*, std::vector< double > & *outAmount*) const

Based on the distribution function and the distance between each cell and the specified center point this function fills the outAmount list with values so that the mean value is 1.

Parameters:
center The coorinate to center the Distribution around.

cells A list of cell positions for which to calculate the fraction.

g The grid that the cells belongs to.

outAmount On return this list has the same size as the cells list and outAmount[i] contains the amount for the cell specified by cells[i].

4.44.3.3 virtual double Distribution::f (double *x*) const [pure virtual]

Gets the value of the distribution at distance x.

Parameters:
x The distance.

Returns:
The value of the distribution at distance x.

Implemented in **CityDistribution** (p. 96), **UniformDistribution** (p. 543), **RandomUniformDistribution** (p. 374), and **NormalDistribution** (p. 305).

4.44.3.4 virtual void Distribution::reset (const DataObject & *d*) [inline, virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

Reimplemented in **CityDistribution** (p. 96), and **NormalDistribution** (p. 306).

4.44.3.5 virtual void Distribution::update (const Update & *u*) [inline, virtual]

Updates this object.

Parameters:
u The **Update**(p. 567) to update this object with.

Implements **SimulationObject** (p. 431).

Reimplemented in **CityDistribution** (p. 97), and **NormalDistribution** (p. 306).

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

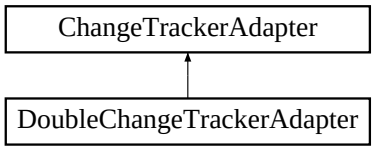
- Distribution.h
- Distribution.cpp

4.45 DoubleChangeTrackerAdapter Class Reference

The DoubleChangeTrackerAdapter keeps track of changes in **StratmasDouble**(p. 464) objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for DoubleChangeTrackerAdapter::



Public Member Functions

- **DoubleChangeTrackerAdapter** (**StratmasDouble** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 177) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **StratmasDouble** & **mObject**

  The adapted **DataObject**(p. 145).

- double **mLast**

  The last value written.

4.45.1 Detailed Description

The DoubleChangeTrackerAdapter keeps track of changes in **StratmasDouble**(p. 464) objects.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.45.2 Constructor & Destructor Documentation

4.45.2.1 DoubleChangeTrackerAdapter::DoubleChangeTrackerAdapter (StratmasDouble & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.45.3 Member Function Documentation

4.45.3.1 bool DoubleChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 177) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 177) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.45.3.2 ostream & DoubleChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.46 EdgeState Struct Reference

Helper struct for the interior finding algorithm.

`#include <AreaHandler.h>`

Public Attributes

- **EdgeState** * **NextEdge**
  The next edge.
- int **X**
  The x-coordinate.
- int **StartY**
  The start y-coordinate.
- int **WholePixelXMove**
  The number of whole pixels to move horizontally for each pixel moved vertically.
- int **XDirection**
  The direction to move in (-1 for left, 1 for right).
- int **ErrorTerm**
  Current error term.
- int **ErrorTermAdjUp**
  Error(p. 205) *term adjustment when moving up.*
- int **ErrorTermAdjDown**
  Error(p. 205) *term adjustment when moving down.*
- int **Count**
  Keeps the count.
- int **EndX**
  For debugging purposes.
- int **EndY**
  For debugging purposes.

4.46.1 Detailed Description

Helper struct for the interior finding algorithm.

Represents a linked list of edges.

Author:
Per Alexius (after Michel Abrash's algorithm).

Date Date

2005/06/15 09:28:18

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

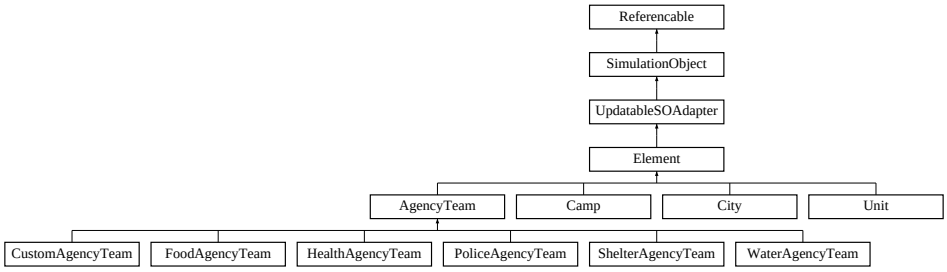
- AreaHandler.h

4.47 Element Class Reference

Abstract base class for a Stratmas Element.

`#include <Element.h>`

Inheritance diagram for Element::



Public Member Functions

- **Element** (const **Reference** &ref, const **Shape** &location)

  Constructor that performs a deep copy of the location. - For camps.

- **Element** (const **DataObject** &d)

  Constructor that creates an Element from a **DataObject**(p. 145).

- virtual ~**Element** ()

  Destructor.

- virtual bool **present** () const =0

  Checks for presence.

- const **Shape** & **location** () const

  Accessor for the location.

- const **Distribution** & **deployment** () const

  Accessor for the distribution.

- **LatLng center** () const

  Accessor for the center coordinate.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- virtual void **replaceObject** (**DataObject** &newObject, int64_t initiator)

  Replaces the **SimulationObject**(p. 429) *with the same reference as the provided* **DataObject**(p. 145) *with a new* **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145).

- virtual void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- virtual void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Protected Attributes

- **Shape** * **mLocation**

  This Element's Location.

- **Distribution** * **mDeployment**

  The distribution of this Element.

4.47.1 Detailed Description

Abstract base class for a Stratmas Element.

Author:
Per Alexius

Date:
2006/02/28 17:48:16

4.47.2 Constructor & Destructor Documentation

4.47.2.1 Element::Element (const Reference & *ref*, const Shape & *location*)

Constructor that performs a deep copy of the location. - For camps.

Parameters:
ref The reference to this Element.
location The location of this Element.

4.47.2.2 Element::Element (const DataObject & *d*)

Constructor that creates an Element from a **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.47.3 Member Function Documentation

4.47.3.1 LatLng Element::center () const

Accessor for the center coordinate.

Returns:
The center coordinate.

4.47.3.2 const Distribution& Element::deployment () const [inline]

Accessor for the distribution.

Returns:
The **Distribution**(p. 173).

4.47.3.3 void Element::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

Reimplemented in **AgencyTeam** (p. 37), **CustomAgencyTeam** (p. 137), **City** (p. 93), and **Unit** (p. 556).

4.47.3.4 const Shape& Element::location () const [inline]

Accessor for the location.

Returns:
The location.

4.47.3.5 void Element::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **UpdatableSOAdapter** (p. 565).

Reimplemented in **AgencyTeam** (p. 37), **CustomAgencyTeam** (p. 137), and **Unit** (p. 557).

4.47.3.6 virtual bool Element::present () const [pure virtual]

Checks for presence.

Returns:
True if this Element is present, false otherwise.

Implemented in **AgencyTeam** (p. 38), **Camp** (p. 75), **City** (p. 93), and **Unit** (p. 558).

4.47.3.7 void Element::replaceObject (DataObject & *newObject*, int64_t *initiator*) [virtual]

Replaces the **SimulationObject**(p. 429) with the same reference as the provided **DataObject**(p. 145) with a new **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145).

Parameters:

newObject The **DataObject**(p. 145) to create the replacing object from.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.47.3.8 void Element::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

Reimplemented in **AgencyTeam** (p. 38), **CustomAgencyTeam** (p. 138), and **Unit** (p. 560).

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

- Element.h
- Element.cpp

4.48 Ellipse Class Reference

Calculates the concentration ellipse for a series of weighted observations of two variables.

`#include <Ellipse.h>`

Public Member Functions

- **Ellipse** ()

  Constructor.

- bool **SetStats** (double inW, double inX, double inY, double inX2, double inY2, double inXY)

  Given input summary statistics, calculates: Weight, mx, my, Vx, Vy, Cxy, Disc.

- bool **SummarizeData** (int n, int which, int *cluster, const double *const xp, const double *const yp, double *wp)

  This function receives a weighted set of n points, and fits a one-sigma ellipse of inertia to the data.

- void **FindEllipse** ()

  Given summary statistics, calculate the parameters of the best-fitting ellipse.

- void **GetSigmaBox** (double k)

  Returns the vertices of a rectangle in general position that contains the k-sigma ellipse of inertia for the data.

- double **Distance2** (double x, double y)

  Returns the Mahalanobis distance of a given point to the center of the ellipse.

- double **ProbDensity** (double x, double y)

  Returns the value of the probability density function of the multivariate normal at the point (x,y).

- double **square** (double x)

  Calculates the square of x.

Public Attributes

- int **nCases**

  Number of observations for this ellipse.

- double **mx**

  x-coordinate of center of the ellipse

- double **my**

  y-coordinate of center of the ellipse

- double **dMajor**

  Half the length of the major axis.

- double **dMinor**

  Half the length of the minor axis.

- double **angle**

  Angle of major axis to the origin.

- double **alpha**

  Arctan of d2/d1 (= 185/4 if circle).

- double **beta**

  First component of first eigenvector.

- double **Vx**

  Variance along x coordinate.

- double **Vy**

  Variance along y coordinate.

- double **Cxy**

  Covariance.

- double **Disc**

  Discriminant.

- double **InvSqrtDisc**

  Inverse of the square root of the discriminant.

- double **Weight**

  Total weight of the ellipse.

- double **vx** [4]

  x-coordinates of the vertex rectangle

- double **vy** [4]

  y-coordinates of the vertex rectangle

4.48.1 Detailed Description

Calculates the concentration ellipse for a series of weighted observations of two variables.

This class is almost identical to its counterpart in older versions of Stratmas. Some name changes have been made in order to match naming conventions in other source code files.

Author:
Loren Cobb - Modified by Per Alexius.

Date **Date**
2005/06/13 11:19:05

4.48.2 Member Function Documentation

4.48.2.1 double Ellipse::Distance2 (double *x*, double *y*)

Returns the Mahalanobis distance of a given point to the center of the ellipse.

Note: units are in squared values!

4.48.2.2 void Ellipse::FindEllipse ()

Given summary statistics, calculate the parameters of the best-fitting ellipse.

Calculates: dMajor, dMinor,angle, alpha, beta

4.48.2.3 double Ellipse::ProbDensity (double *x*, double *y*) [inline]

Returns the value of the probability density function of the multivariate normal at the point (x,y).

Parameters:

x x-coordinate

y y-coordinate

Returns:

The value of the probability density function of the multivariate normal at the point (x,y).

4.48.2.4 bool Ellipse::SetStats (double *inW*, double *inX*, double *inY*, double *inX2*, double *inY2*, double *inXY*)

Given input summary statistics, calculates: Weight, mx, my, Vx, Vy, Cxy, Disc.

Parameters:

inW Sum of weights

inX Sum of x-coords

inY Sum of y-coords

inX2 Sum of squared x-coords

inY2 Sum of squared y-coords

inXY Sum of x-y coord products

4.48.2.5 bool Ellipse::SummarizeData (int *n*, int *which*, int * *cluster*, const double *const *xp*, const double *const *yp*, double * *wp*)

This function receives a weighted set of n points, and fits a one-sigma ellipse of inertia to the data.

If the data are bivariate normal, then the ellipse of inertia will contain 67% of the data points.

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

- Ellipse.h
- Ellipse.cpp

4.49 EnemyRecord Class Reference

Record for holding information of damage in different cells.

`#include <Unit.h>`

Public Member Functions

- **EnemyRecord** ()

  Default constructor.

- **EnemyRecord** (const **EnemyRecord** &er)

  Copy constructor.

- double **damage** () const

  Accessor for the damage.

- const std::set< int > & **cells** () const

  Accessor for the set of cells this record refers to.

- void **addCell** (int cell)

  Adds a cell to this record.

- void **addDamage** (double damage)

  Adds some damage to this record.

Private Attributes

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

  The set of cells the damage is inflicted in.

- double **mDamage**

  The total damage in the cells this record refers to.

4.49.1 Detailed Description

Record for holding information of damage in different cells.

Author:
Per Alexius

Date:
2006/04/21 15:54:52

4.49.2 Constructor & Destructor Documentation

4.49.2.1 EnemyRecord::EnemyRecord (const EnemyRecord & *er*) [inline]

Copy constructor.

Parameters:
er The record to copy.

4.49.3 Member Function Documentation

4.49.3.1 void EnemyRecord::addCell (int *cell*) [inline]

Adds a cell to this record.

Parameters:
cell The index (in the active array) of the cell to add.

4.49.3.2 void EnemyRecord::addDamage (double *damage*) [inline]

Adds some damage to this record.

Parameters:
damage The damage to add.

4.49.3.3 const std::set<int>& EnemyRecord::cells () const [inline]

Accessor for the set of cells this record refers to.

Returns:
The set of cells this record refers to.

4.49.3.4 double EnemyRecord::damage () const [inline]

Accessor for the damage.

Returns:
The damage.

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

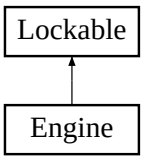
- Unit.h

4.50 Engine Class Reference

This class represents the 'engine' that runs the simulation.

`#include <Engine.h>`

Inheritance diagram for Engine::



Public Member Functions

- **Engine** (**Buffer** &b)

  Constructor.

- ~**Engine** ()

  Destructor.

- bool **initialized** () const

  Accessor for the mInitialized flag.

- void **createSimulation** (**DataObject** *simulation, int64_t creator)

  Ends current simulation (if any) and creates a new one base on the provided **DataObject**(p. 145).

- void **setNumberOfTimesteps** (int ts)

  Mutator for the number of timesteps.

- **EngineStatusObject wait** ()

  Waits for the Engine to finish the task it is currently performing.

- void **run** ()

  The main loop of the Engine.

- void **put** (enum eEngMsg msg)

  Sends a command to the Engine.

- **UniqueTime registerInterestInTime** (**Time** t, **TSQueue**< **EngineStatusObject** > *q)

  Notifies the Engine that a passive client is interested in getting data for a specified time step.

- void **deregisterInterestInTime** (**UniqueTime** ut)

  Notifies the Engine that a passive client is no longer interested in getting data for a specified time step.

Static Public Member Functions

- void * **start** (void *engineToStart)

  Starts the specified engine.

Private Member Functions

- void **endSimulation** ()

  Ends the current **Simulation**(p. 424).

- void **notifyAllTimeListeners** (const char *errMsg=0)

  Notifies all time listeners that something has happend that may have changed the simulation so that an **UpdateMessage**(p. 570) *should be sent.*

Private Attributes

- **Buffer** & **mBuf**

  A reference to the **Buffer**(p. 67).

- **TSQueue**< int > **mQ**

  The queue used by other threads to communicate with the Engine thread.

- **TSQueue**< **EngineStatusObject** > **mOutQ**

  The queue used by the Engine to communicate with the active client.

- std::map< **UniqueTime**, **TSQueue**< **EngineStatusObject** > * > **mTimeListeners**

  Maps a **UniqueTime**(p. 544) *to the queue used for communicating with the thread representing for* **Session**(p. 408) *that is interested in that time.*

- bool **mInitialized**

  Indicates if the simulation is initialized or not.

- **Simulation** * **mSimulation**

  A pointer to the **Simulation**(p. 424).

- int **mNumberOfTimesteps**

  Number of steps to iterate when getting the next step command.

4.50.1 Detailed Description

This class represents the 'engine' that runs the simulation.

The engine listens to commands from a **Session**(p. 408) and acts according to those commands. The thread running the Engine main loop is the thread that performs the actual simulation work.

Author:
Per Alexius

Date Date
2006/03/06 09:18:09

4.50.2 Constructor & Destructor Documentation

4.50.2.1 Engine::Engine (Buffer & *b*)

Constructor.

Parameters:
b The **Buffer**(p. 67).

4.50.3 Member Function Documentation

4.50.3.1 void Engine::createSimulation (DataObject * *simulation*, int64_t *creator*)

Ends current simulation (if any) and creates a new one base on the provided **DataObject**(p. 145).

Parameters:
simulation The **DataObject**(p. 145) to create the simulation from.
creator The id of the **Session**(p. 408) creating the simulation.

4.50.3.2 void Engine::deregisterInterestInTime (UniqueTime *ut*)

Notifies the Engine that a passive client is no longer interested in getting data for a specified time step.

Called by **Session**(p. 408) objects representing passive clients when using a polling strategy for detecting if any new time steps have been calculated.

Parameters:
ut The **UniqueTime**(p. 544) received when registering interest in the time we now want to deregister interest in.

4.50.3.3 bool Engine::initialized () const [inline]

Accessor for the mInitialized flag.

Returns:
The status of the mInitialized flag.

4.50.3.4 void Engine::put (enum eEngMsg *msg*)

Sends a command to the Engine.

Parameters:
msg The command to send according to the eEngMsg enumeration.

4.50.3.5 UniqueTime Engine::registerInterestInTime (Time *t*, TSQueue< EngineStatusObject > * *q*)

Notifies the Engine that a passive client is interested in getting data for a specified time step.

Called by **Session**(p. 408) objects representing passive clients when receiving a StepMessage. The simulation time asked for may or may not have passed.

Parameters:

t The simulation time the passive client is interested in getting data for.

q The **TSQueue**(p. 526) the Engine should use when notifying the passive client that the data is available.

Returns:

A **UniqueTime**(p. 544) that may be used later in order to deregister interest in this time.

4.50.3.6 void Engine::run ()

The main loop of the Engine.

Blocks and waits for commands from the **Session**(p. 408) representing the active client. When receiving a command the appropriate actions will be taken.

4.50.3.7 void Engine::setNumberOfTimesteps (int *ts*) [inline]

Mutator for the number of timesteps.

Parameters:

ts The number of timesteps.

4.50.3.8 void * Engine::start (void * *engineToStart*) [static]

Starts the specified engine.

Used as start function when creating the engine thread.

Parameters:

engineToStart The Engine instance to start.

Returns:

Null if the tread exited successfully, undefined otherwise.

4.50.3.9 EngineStatusObject Engine::wait () [inline]

Waits for the Engine to finish the task it is currently performing.

Blocks the calling thread until the task is performed. Called by the **Session**(p. 408) representing the active client for example when waiting for an initializatio or a time step to be finished.

Returns:

The status of the performed task according to the eOutMsg enumeration.

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

- Engine.h
- Engine.cpp

4.51 EngineStatusObject Class Reference

Object returned by the **Engine**(p. 189) when a task enqueued by a **Session**(p. 408) is finished.

`#include <Engine.h>`

Public Member Functions

- **EngineStatusObject** ()

  Default constructor.

- **EngineStatusObject** (const **Error** &e)

  Creates a status object containing an **Error**(p. 205).

- **EngineStatusObject** (const std::vector< **Error** > &e)

  Creates a status object containing multiple Errors.

- **EngineStatusObject** (const **EngineStatusObject** &e)

  Copy constructor.

- bool **errorOccurred** () const

  Checks if this status object contains any errors.

- std::vector< **Error** > **errors** () const

  Accessor for the error vector.

- **EngineStatusObject** & **operator**= (const **EngineStatusObject** &e)

  Assignment operator.

Private Attributes

- std::vector< **Error** > **mErrors**

  Vector holding the errors.

4.51.1 Detailed Description

Object returned by the **Engine**(p. 189) when a task enqueued by a **Session**(p. 408) is finished.

Author:
Per Alexius

Date **Date**
2006/03/06 09:18:09

4.51.2 Constructor & Destructor Documentation

4.51.2.1 EngineStatusObject::EngineStatusObject (const Error & *e*) [inline]

Creates a status object containing an **Error**(p. 205).

Parameters:
e The error.

4.51.2.2 EngineStatusObject::EngineStatusObject (const std::vector< Error > & *e*) [inline]

Creates a status object containing multiple Errors.

Parameters:
e A vector containing the errors.

4.51.2.3 EngineStatusObject::EngineStatusObject (const EngineStatusObject & *e*) [inline]

Copy constructor.

Parameters:
e The object to copy.

4.51.3 Member Function Documentation

4.51.3.1 bool EngineStatusObject::errorOccurred () const [inline]

Checks if this status object contains any errors.

Returns:
True if this status object contains any errors.

4.51.3.2 std::vector<Error> EngineStatusObject::errors () const [inline]

Accessor for the error vector.

Returns:
The error vector.

4.51.3.3 EngineStatusObject& EngineStatusObject::operator= (const EngineStatusObject & *e*) [inline]

Assignment operator.

Parameters:
e The object to assign from.

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

- Engine.h

4.52 Environment Class Reference

Helper class for keeping track of some environment related variables.

`#include <Environment.h>`

Static Public Member Functions

- std::string **getNativePath** (const std::string &path)

  Gets the absolute path to the given filename. The following implicit conventions apply:.

- std::string **getDumpDir** ()
- void **milliSleep** (int secs)

  Suspends the calling thread for the specified number of milliseconds.

- std::string **getVersion** ()

  Accessor for the Stratmas version.

- void **initEnvironment** (int argc, char **argv)

  Initializes and configures the environment, exits on error. Also handles (via initConfiguration) and exits on version or help queries. This function can be called several times, however it will immidiatelly return on all but the first invocation. Currently the arbitration of "firstness" is not thread-safe, in practice this should not currently be a problem.

- std::string **getProgramName** ()
- int **getServerPort** ()
- const std::string & **getServerAddress** ()
- **ClientValidator** * **getClientValidator** ()
- bool **getUseDispatcher** ()
- const std::string & **getDispatcherHost** ()
- int **getDispatcherPort** ()

Static Public Attributes

- const std::string **DEFAULT_SCHEMA_NAMESPACE**

  The default namespace.

- const std::string **XSD_NAMESPACE**

  The xsd namespace.

- const std::string **STRATMAS_PROTOCOL_SCHEMA**

  The name of the main Stratmas schema file.

Static Private Member Functions

- void **initInstallDir** (const fs::path &executable)

  Tries to determine the install dir, and dependent paths. Currently it is necessary to hint the method with the path to the executable.

- void **initConfig** (int argc, char **argv)

  Configures the environment, exits on error. Also handles (and exits) on version or help queries. initInstallDir is assumed to have been called prior to this function.

- void **setConfigFile** (const std::string &file)

  Sets the path to the config file, an important side effect of this function is that it also sets variables to the effect that _not_ finding the configuration file is an error.

- void **setDumpDir** (const std::string &dirname)

  Sets the directory where stratmas will dump files, also activates file dumping.

- void **addPlatformOptions** (po::options_description *invocation, po::options_description *configuration, po::options_description *development)

  Fills in platform specific options, which in the posix case are the daemonization option.

- void **handlePlatformOptions** (po::variables_map *vm)

  Callback to handle platform options specified in addPlatformOptions.

- const fs::path & **getInstallDir** ()

  Returns the directory where stratmas is assumed to be installed.

- std::string **getNativePath** (const fs::path &path)

  Gets the absolute path to the given filename.

- std::string **getInstalledPath** (const fs::path &installedPath)
- fs::path **importNativePath** (const std::string &nativePath)

  tries to parse the provided string as a os-native path and returns the stratmas native path representation. The following implicit conventions apply:

- bool **isFile** (const fs::path &path)

  File tester.

Static Private Attributes

- const std::string **STRATMAS_VERSION**

  CVS will replace this string with the current version number.

- const std::string **DEFAULT_VERSION**

  Version string for versions with no version number.

- bool **initStarted** = false
- fs::path **sExecutable**

  The path to the program.

- fs::path **sInstallDir**

  The installation directory.

- fs::path **sConfigFile**

  Path to the configuration file.

- fs::path **sDumpDir**

  The folder to write files to.

- bool **sForceConfigFileRead** = false

  If set to true, the lack of a config file is an error.

- std::string **sServerAddress**

  The server address.

- int **sServerPort** = 28444

  The server port.

- bool **useDispatcher** = false

  If a dispatcher should be used.

- std::string **sDispatcherHost**

  The dispatcher address.

- int **sDispatcherPort** = 4181

  The dispatcher port.

- **ClientValidator** * **spClientValidator** = new **PassClientValidator**()

4.52.1 Detailed Description

Helper class for keeping track of some environment related variables.

Author:
Per Alexius

Date **Date**
2006/09/11 09:00:30

4.52.2 Member Function Documentation

4.52.2.1 std::string Environment::getDumpDir () [static]

Returns the directory designated for dumping debug files in.

4.52.2.2 const fs::path & Environment::getInstallDir () [static, private]

Returns the directory where stratmas is assumed to be installed.

Returns:
the directory where stratmas is assumed to be installed

4.52.2.3 std::string Environment::getNativePath (const std::string & *path*) [static]

Gets the absolute path to the given filename. The following implicit conventions apply:.

1. If the filename is absolute (=complete) the native representation will be provided.

2. If the filename is relative it is resolved relative to **getInstallDir()**(p. 199) which currently is the directory where the executable lives (that may change) and the native absolute path is returned.

Returns:
The absolute path to the given filename.

4.52.2.4 std::string Environment::getNativePath (const fs::path & *path*) [static, private]

Gets the absolute path to the given filename.

Returns:
The absolute path to the given filename.

4.52.2.5 std::string Environment::getProgramName () [static]

Returns the name of the program

4.52.2.6 string Environment::getVersion () [static]

Accessor for the Stratmas version.

Returns:
The Stratmas version.

4.52.2.7 fs::path Environment::importNativePath (const std::string & *nativePath*) [static, private]

tries to parse the provided string as a os-native path and returns the stratmas native path representation. The following implicit conventions apply:

1. If the filename is absolute (=complete) it is stored absolute. 2. If the filename is relative it is resolved relative to **getInstallDir()**(p. 199) which currently is the directory where the executable lives (that may change).

TODO proper errorhandling.

4.52.2.8 bool Environment::isFile (const fs::path & *path*) [static, private]

File tester.

Returns:
returns true if the provided path points to an existing non-directory.

4.52.2.9 void Environment::setConfigFile (const std::string & *file*) [static, private]

Sets the path to the config file, an important side effect of this function is that it also sets variables to the effect that _not_ finding the configuration file is an error.

As a temporary solution of the difficulties involving finding out if this value was applied as a default or not, setting the config file to the empty is a noop.

4.52.3 Member Data Documentation

4.52.3.1 bool Environment::initStarted = false [static, private]

If environment initialization has started. Currently this is mainly used to ensure that a subsequent calls to main when using Windows * S ervices won't try to reinitialize the environment.

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

- Environment.h
- Environment.cpp
- optionsPosix.cpp
- optionsWin.cpp

4.53 EpidemicsWeights Class Reference

Static class for precalculating expensive exp() and sqrt() functions used in the 'AttrFractionInfected' attribute.

`#include <EpidemicsWeights.h>`

Static Public Member Functions

- void **setEpidemicsWeights** (double meanContact, double maxContact, double cellSideMeters, int nrows, int ncols)

  Initialize the weights. Called only by **Grid**(p. 227) *constructor.*

- void **limits** (int row, int col, int &top, int &left, int &bottom, int &right)

  Returns top left and bottom right row and column for the area of influence measured from 'row' and 'col'.

- double **weight** (int i)

  Returns weight 'i' where 'i' is the number of the cell in the epidemic's area of influence counted from top left to bottom right.

- void **clear** ()

  Free memory. Called only by **Grid**(p. 227) *destructor.*

Static Private Attributes

- int **mRows** = 0

  Number of rows in the **Grid**(p. 227).

- int **mCols** = 0

  Number of colums in the **Grid**(p. 227).

- int **mHalfNumberOfCells** = 0

  Radius of the epidemic's area of influence measured in in cells.

- double **mCellSideMeters** = 0

  Size of the cells.

- double * **mEpidemicsWeight** = 0

  Vector containing the weights.

4.53.1 Detailed Description

Static class for precalculating expensive exp() and sqrt() functions used in the 'AttrFractionInfected' attribute.

The 'setEpidemicsWeights()' function is called from the Grid's constructor which guarantees that the static members are initialized with reasonable values when the simulation starts.

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

- EpidemicsWeights.h
- GridCellPV.cpp

4.54 equalGridCellPtr Struct Reference

Function object for equality operator for pointer to GridCells.

`#include <GridCell.h>`

Public Member Functions

- bool **operator()** (const **GridCell** *c1, const **GridCell** *c2)

  Equality operator for pointers to GridCells.

4.54.1 Detailed Description

Function object for equality operator for pointer to GridCells.

Two cells are equal if their row and column number matches.

Author:
Per Alexius

Date:
2007/01/28 17:07:49

4.54.2 Member Function Documentation

4.54.2.1 bool equalGridCellPtr::operator() (const GridCell * *c1*, const GridCell * *c2*) [inline]

Equality operator for pointers to GridCells.

Parameters:
c1 The first cell.

c2 The second cell.

Returns:
True if the two cells are equal.

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

- GridCell.h

4.55 Error Class Reference

This class represents an error that the server has found and that is fatal for the currently ongoing simulation. If a client receives an Error the currently ongoing simulation should not be trusted.

`#include <Error.h>`

Public Types

- enum **eType** { **eWarning**, **eGeneral**, **eFatal** }
  Enumeration for Error types.

Public Member Functions

- **Error** (int type=eGeneral)
  Default constructor that creates an Error with an empty message;.
- **Error** (const char *s, int type=eGeneral)
  Consructor that creates an Error with the specified message.
- **Error** (const **Error** &e)
  Copy consructor.
- **Error** & **operator=** (const **Error** &e)
  Assignment operator.
- int **type** () const
  Accessor for the Error type.
- std::string **typeStr** () const
  Gets the type of this error as a string.
- template<class T> **Error** & **operator<<** (T t)
  Writes the provided object to this Error's message stream.
- **Error** & **operator<<** (const **Reference** &ref)
  Writes the provided **Reference**(p. 378) *to this Error's message stream.*
- void **toXML** (std::ostream &o) const
  Produces the XML representation of this object.

Private Attributes

- int **mType**
  The type of the Error according to the eType enumeration.
- std::ostringstream **mMsg**
  A message describing the error.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Error** &e)

  For debugging purposes.

4.55.1 Detailed Description

This class represents an error that the server has found and that is fatal for the currently ongoing simulation. If a client receives an Error the currently ongoing simulation should not be trusted.

Author:
Per Alexius

Date:
2006/03/06 12:55:09

4.55.2 Constructor & Destructor Documentation

4.55.2.1 Error::Error (int *type* = eGeneral) [inline]

Default constructor that creates an Error with an empty message;.

Parameters:
type The type of Error to create.

4.55.2.2 Error::Error (const char * *s*, int *type* = eGeneral) [inline]

Consructor that creates an Error with the specified message.

Parameters:
s The message describing the error.
type The type of Error to create.

4.55.2.3 Error::Error (const Error & *e*) [inline]

Copy consructor.

Parameters:
e The Error to copy.

4.55.3 Member Function Documentation

4.55.3.1 Error & Error::operator<< (const Reference & *ref*)

Writes the provided **Reference**(p. 378) to this Error's message stream.

Parameters:
ref The **Reference**(p. 378) to write.

Returns:
A reference to this Error.

4.55.3.2 template<class T> Error& Error::operator<< (T *t*) [inline]

Writes the provided object to this Error's message stream.

Parameters:
t The object to write.

Returns:
A reference to this Error.

4.55.3.3 Error& Error::operator= (const Error & *e*) [inline]

Assignment operator.

Parameters:
e The Error which value to assign to this Error.

Returns:
This Error with its new value.

4.55.3.4 void Error::toXML (std::ostream & *o*) const

Produces the XML representation of this object.

Parameters:
o The stream to write the XML representation to.

4.55.3.5 int Error::type () const [inline]

Accessor for the Error type.

Returns:
The type of this error.

4.55.3.6 string Error::typeStr () const

Gets the type of this error as a string.

Returns:
The type of this error as a string.

4.55.4 Friends And Related Function Documentation

4.55.4.1 std::ostream& operator<< (std::ostream & *o*, const Error & *e*) [friend]

For debugging purposes.

Parameters:

o The stream to write to.

e The Error to print.

Returns:

The provided ostream with the Error written to it.

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

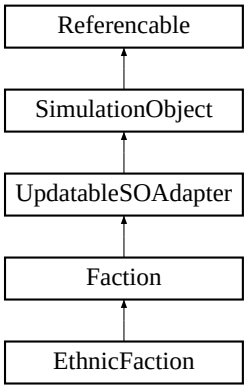
- Error.h
- Error.cpp

4.56 EthnicFaction Class Reference

The EthnicFaction class contains the Stratmas server representation of an EthnicFaction.

`#include <Faction.h>`

Inheritance diagram for EthnicFaction::



Public Member Functions

- **EthnicFaction** (const **DataObject** &d)

  Constructor that creates an EthnicFaction from a **DataObject**(p. 145).

- virtual ~**EthnicFaction** ()

  Destructor.

- bool **isAll** () const

  Returns true if this faction is the all faction.

- int **index** () const

  Returns the index of this faction.

Static Public Member Functions

- **EthnicFaction** & **all** ()

  Gets the 'all' faction. If it does not already exist it is created.

- **EthnicFaction** * **faction** (int i)

  Maps a Faction's index to the faction itself.

- **EthnicFaction** * **faction** (const **Reference** &ref)

  Maps an EthnicFaction's **Reference**(p. 378) *to the faction itself.*

Static Public Attributes

- const int **ALL** = 0

  The index of the all faction.

- const int **NONE** = -1

  Represents no faction.

Private Member Functions

- **EthnicFaction** ()

  Private default constructor for the all faction.

Static Private Attributes

- **EthnicFaction** * **sAllFaction**

  Pointer to the 'all' faction.

- int **sCurrentIndex**

  The index of the next faction to create.

- std::map< int, **EthnicFaction** * > **mFactionIndexMap**

  Maps a Faction's index to the faction itself. Should not contain the all faction.

4.56.1 Detailed Description

The EthnicFaction class contains the Stratmas server representation of an EthnicFaction.

Author:
Per Alexius

Date:
2006/07/05 14:49:43

4.56.2 Constructor & Destructor Documentation

4.56.2.1 EthnicFaction::EthnicFaction (const DataObject & *d*)

Constructor that creates an EthnicFaction from a **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.56.3 Member Function Documentation

4.56.3.1 EthnicFaction & EthnicFaction::all () [static]

Gets the 'all' faction. If it does not already exist it is created.

Returns:
The 'all' faction.

4.56.3.2 EthnicFaction* EthnicFaction::faction (const Reference & *ref*) [inline, static]

Maps an EthnicFaction's **Reference**(p. 378) to the faction itself.

Parameters:
ref The **Reference**(p. 378) to the ethnic faction.

Returns:
A pointer to the ethnic faction with **Reference**(p. 378) ref or null of no such ethnic faction exists.

Reimplemented from **Faction** (p. 214).

4.56.3.3 EthnicFaction* EthnicFaction::faction (int *i*) [inline, static]

Maps a Faction's index to the faction itself.

Parameters:
i The index of the faction.

Returns:
A pointer to the faction with index i of null of no such faction exists.

4.56.3.4 int EthnicFaction::index () const [inline]

Returns the index of this faction.

Returns:
The index of this faction.

4.56.3.5 bool EthnicFaction::isAll () const [inline]

Returns true if this faction is the all faction.

Returns:
True if this faction is the all faction, false otherwise.

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

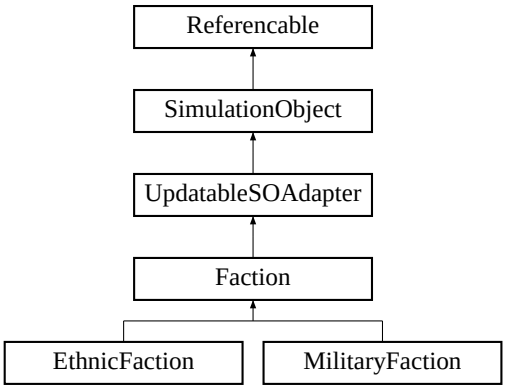
- Faction.h
- Faction.cpp

4.57 Faction Class Reference

The Faction class is the abstract base class for the Stratmas server representation of differennt types of Factions.

`#include <Faction.h>`

Inheritance diagram for Faction::



Public Member Functions

- **Faction** (const **DataObject** &d)
- virtual ~**Faction** ()
- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- void **replaceObject** (**DataObject** &newObject, int64_t initiator)

  Replaces the **SimulationObject**(p. 429) *with the same reference as the provided* **DataObject**(p. 145) *with a new* **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145).

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- bool **isHostileTowards** (const **Faction** &f)

  Checks if this faction is hostile towards the specified faction.

Static Public Member Functions

- **Faction** * **faction** (const **Reference** &ref)

  Maps a Faction's **Reference**(p. 378) *to the faction itself.*

Protected Member Functions

- **Faction** ()

  Protected constructor for the All faction.

Protected Attributes

- int **mIndex**

  The index of this Faction. Always 0 for the Faction representing everyone and -1 for other than EthnicFactions.

Private Member Functions

- void **createEnemySet** ()

  Stores the enemies of this faction in the mEnemySet. Necessary since two different entries in the enemy list may refer to the same faction.

Private Attributes

- std::map< const **Reference** *, const **Reference** * > **mEnemyList**

  Maps References to the position in the enemy list to the FactionReference to that faction.

- RefSet **mEnemy**

  References to the factions that this faction consider as its enemies.

Static Private Attributes

- RefFactionMap **mFactionRefMap**

  Maps a **Reference**(p. 378) *to its Faction.*

4.57.1 Detailed Description

The Faction class is the abstract base class for the Stratmas server representation of differennt types of Factions.

Author:
Per Alexius

Date Date
2006/07/05 14:49:43

4.57.2 Constructor & Destructor Documentation

4.57.2.1 Faction::Faction (const DataObject & *d*)

Constructor that creates a Faction from a **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.57.2.2 Faction::~Faction () [virtual]

Destructor.

4.57.3 Member Function Documentation

4.57.3.1 void Faction::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:
toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.
initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 564).

4.57.3.2 void Faction::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.57.3.3 Faction* Faction::faction (const Reference & *ref*) [inline, static]

Maps a Faction's **Reference**(p. 378) to the faction itself.

Parameters:
ref The **Reference**(p. 378) to the faction.

Returns:
A pointer to the faction with **Reference**(p. 378) ref or null of no such faction exists.

Reimplemented in **EthnicFaction** (p. 211).

4.57.3.4 bool Faction::isHostileTowards (const Faction & *f*) [inline]

Checks if this faction is hostile towards the specified faction.

Parameters:
f The faction to check for hostility towards.

Returns:
True if the factions are enemies, false otherwise.

4.57.3.5 void Faction::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.57.3.6 void Faction::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:
toRemove The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.57.3.7 void Faction::replaceObject (DataObject & *newObject*, int64_t *initiator*) [virtual]

Replaces the **SimulationObject**(p. 429) with the same reference as the provided **DataObject**(p. 145) with a new **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145).

Parameters:
newObject The **DataObject**(p. 145) to create the replacing object from.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.57.3.8 void Faction::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

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

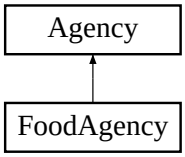
- Faction.h
- Faction.cpp

4.58 FoodAgency Class Reference

Class containing functionality for controlling FoodAgencyTeams.

`#include <Agency.h>`

Inheritance diagram for FoodAgency::



Public Member Functions

- **FoodAgency** (const std::vector< **AgencyTeam** * > &teams, **Grid** &g)

  Constructor.

Private Member Functions

- bool **severeProblem** ()

  Determines if we have a severe resource problem. If we do - then the weights for the clustering algorithm is set.

4.58.1 Detailed Description

Class containing functionality for controlling FoodAgencyTeams.

Author:
Per Alexius

Date:
2006/10/02 16:01:25

4.58.2 Constructor & Destructor Documentation

4.58.2.1 FoodAgency::FoodAgency (const std::vector< AgencyTeam * > & *teams*, Grid & *g*) [inline]

Constructor.

Parameters:
teams A vector containing this Agency's teams.

g A reference to the **Grid**(p. 227).

4.58.3 Member Function Documentation

4.58.3.1 bool FoodAgency::severeProblem () [private, virtual]

Determines if we have a severe resource problem. If we do - then the weights for the clustering algorithm is set.

Returns:
True if we have a severe food problem, false otherwise.

Implements **Agency** (p. 28).

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

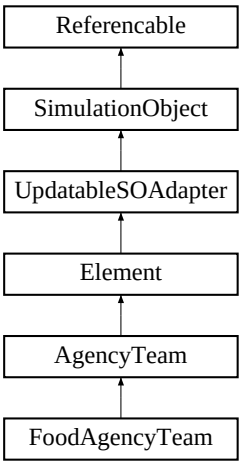
- Agency.h
- Agency.cpp

4.59 FoodAgencyTeam Class Reference

Class representing a FoodAgencyTeam.

`#include <AgencyTeam.h>`

Inheritance diagram for FoodAgencyTeam::



Public Member Functions

- **FoodAgencyTeam** (const **DataObject** &d)

  Constructor that creates an **AgencyTeam**(p. 32) *from the provided* **DataObject**(p. 145).

- double **calculateNeed** ()

  Calculates food need among the population in this team's area of influence.

- void **act** (**Time** now)

  Distribute food in this team's area of influence according to its capacity.

4.59.1 Detailed Description

Class representing a FoodAgencyTeam.

Author:
Per Alexius

Date **Date**
2006/10/10 09:35:59

4.59.2 Constructor & Destructor Documentation

4.59.2.1 FoodAgencyTeam::FoodAgencyTeam (const DataObject & *d*) [inline]

Constructor that creates an **AgencyTeam**(p. 32) from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.59.3 Member Function Documentation

4.59.3.1 void FoodAgencyTeam::act (Time *now*) [virtual]

Distribute food in this team's area of influence according to its capacity.

Parameters:
now The current simulation time.

Implements **AgencyTeam** (p. 35).

4.59.3.2 double FoodAgencyTeam::calculateNeed () [virtual]

Calculates food need among the population in this team's area of influence.

Food need is calculated as follows: For all cells in this team's area of influence with population p $>$ 0.5 persons where the amount of stored food f is less than one day's consumption - add (1 - f) $*$ p to the total need.

Returns:
The need for food in this team's area of influence represented as person-days.

Reimplemented from **AgencyTeam** (p. 36).

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

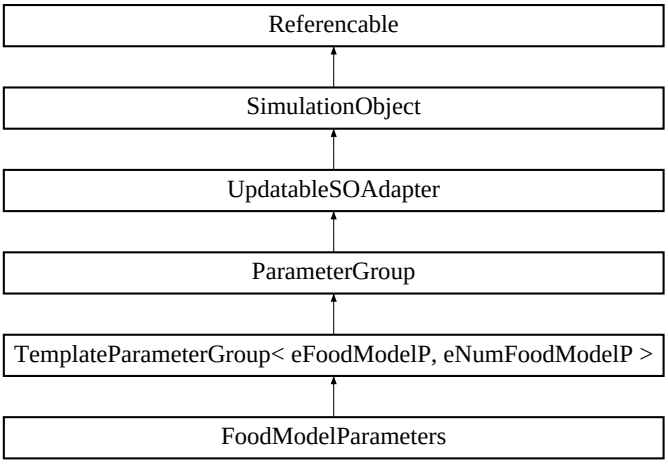
- AgencyTeam.h
- AgencyTeam.cpp

4.60 FoodModelParameters Class Reference

The food model parameter group. This refers to a model that isn't implemented yet. It has been left here for future use.

`#include <ValidParameterGroups.h>`

Inheritance diagram for FoodModelParameters::



Public Member Functions

- **FoodModelParameters** (const **DataObject** &d)

4.60.1 Detailed Description

The food model parameter group. This refers to a model that isn't implemented yet. It has been left here for future use.

Author:
Per Alexius

Date:
2007/01/24 13:13:25

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

- ValidParameterGroups.h

4.61 GaussSaver Class Reference

Helper class for storing info of the number that is saved by the gaussian random number algorithm.

`#include <random.h>`

Static Public Member Functions

- void **setSaved** (bool saved)
  Mutator for the saved flag.
- bool **isSaved** ()
  Accessor for the saved flag.

Static Private Attributes

- bool **smSaved** = false
  Indicates wheter a number is saved or not.

4.61.1 Detailed Description

Helper class for storing info of the number that is saved by the gaussian random number algorithm.

Author:
Per Alexius

Date Date
2006/09/05 14:18:21

4.61.2 Member Function Documentation

4.61.2.1 bool GaussSaver::isSaved () `[inline, static]`

Accessor for the saved flag.

Returns:
The new state of the saved flag.

4.61.2.2 void GaussSaver::setSaved (bool *saved*) `[inline, static]`

Mutator for the saved flag.

Parameters:
saved The new state of the saved flag.

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

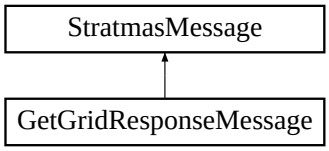
- **random.h**
- **stratmas.cpp**

4.62 GetGridResponseMessage Class Reference

Class representing the GetGridResponseMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for GetGridResponseMessage::



Public Member Functions

- **GetGridResponseMessage** (const **Buffer** &buf, bool bigEndian)
  Constructor.
- void **toXML** (std::ostream &o) const
  Produces the XML representation of this message.

Private Attributes

- const **Buffer** & **mBuf**
  The **Buffer**(p. 67) *to fetch data from.*
- bool **mSessionBigEndian**
  Indicates the byte order of the client receiving the message.

4.62.1 Detailed Description

Class representing the GetGridResponseMessage.

Author:
Per Alexius

Date **Date**
2006/03/06 14:23:12

4.62.2 Constructor & Destructor Documentation

4.62.2.1 GetGridResponseMessage::GetGridResponseMessage (const Buffer & *buf*, bool *bigEndian*) [inline]

Constructor.

Parameters:
buf The **Buffer**(p. 67) to fetch data from, if necessary.
bigEndian Indicates the byte order of the client receiving the message

4.62.3 Member Function Documentation

4.62.3.1 void GetGridResponseMessage::toXML (std::ostream & *o*) const [virtual]

Produces the XML representation of this message.

Parameters:

o The stream to which the message is written

Implements **StratmasMessage** (p. 473).

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

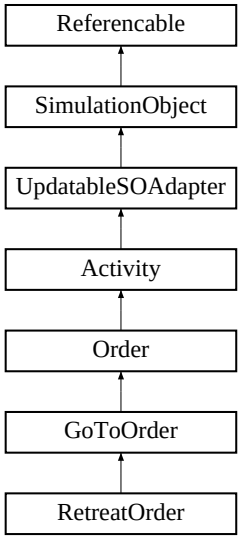
- StratmasMessage.h
- StratmasMessage.cpp

4.63 GoToOrder Class Reference

The GoToOrder.

`#include <Activity.h>`

Inheritance diagram for GoToOrder::



Public Member Functions

- **GoToOrder** (const **Shape** &location)

  Constructor used internally by the server.

- **GoToOrder** (const **DataObject** &d)

  Creates a GoToOrder from the provided **DataObject**(p. 145).

- virtual void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

- double **combatFactor** () const

  Accessor for the combat factor.

4.63.1 Detailed Description

The GoToOrder.

Author:
Per Alexius

Date **Date**
2006/07/19 07:04:26

4.63.2 Constructor & Destructor Documentation

4.63.2.1 GoToOrder::GoToOrder (const Shape & *location*)

Constructor used internally by the server.

Parameters:
location The location to go to.

4.63.2.2 GoToOrder::GoToOrder (const DataObject & *d*) [inline]

Creates a GoToOrder from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.63.3 Member Function Documentation

4.63.3.1 double GoToOrder::combatFactor () const [inline, virtual]

Accessor for the combat factor.

Returns:
The combat factor.

Implements **Order** (p. 310).

4.63.3.2 void GoToOrder::perform (Element * *e*, double *fraction* = 1.0) [virtual]

Performs this **Activity**(p. 21).

Parameters:
e The **Element**(p. 180) that should perform this **Activity**(p. 21).
fraction The fraction of the performers total capacity that this activity is performed with.

Implements **Activity** (p. 23).

Reimplemented in **RetreatOrder** (p. 395).

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

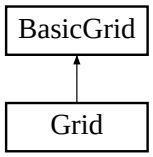
- Activity.h
- Activity.cpp

4.64 Grid Class Reference

This class represents the simulation grid.

`#include <Grid.h>`

Inheritance diagram for Grid::



Public Member Functions

- **Grid** (const **Map** &amap, double cellSizeMeters, int nEthnic)
  Creates a Grid for the provided **Map**(p. 292) *with the specified cell size.*
- virtual ~**Grid** ()
  Destructor.
- int **factions** () const
  Accessor for the number of factions, excluding the 'all' faction.
- int **camps** () const
  Accessor for the number of camps.
- **Camp** * **camp** (unsigned int i) const
  Accessor for camps.
- double **HDI** () const
  Accessor for the HDI parameter.
- double **unemployment** () const
  Accessor for the unemployment parameter.
- double **regionFoodSurplus** () const
- double **regionFoodDeficit** () const
- void **updateRegions** (const std::vector< **Region** * > &regions)
- double **totalPopulation** () const
  Accessor for the total population in the entire grid.
- double **totalInitialPopulation** (int f=0) const
  Accessor for the initial population of a faction in the entire grid.
- double **initialPopulation** (int i, int f=0) const
  Accessor for the initial population of a faction in a cell.
- double **resettlers** (int f) const

- const double *const **cellCenterCoordsX** () const

  Accessor for the cell center x coordinate array used for clustering.

- const double *const **cellCenterCoordsY** () const

  Accessor for the cell center y coordinate array used for clustering.

- const **CombatGrid** * **cg** () const

  Accessor for the cell center y coordinate array used for clustering.

- **GridCell** * **cell** (int ind) const

  Gets the cell with the specified index in the active cells array.

- **GridCell** * **cell** (int r, int c)

  Gets the cell with the specified row and column number.

- **GridCell** * **cell** (const **GridPos** &p)

  Gets the cell at the specified grid position.

- const **GridCell** * **cell** (int r, int c) const

  Gets the cell with the specified row and column number.

- const **GridCell** * **cell** (const **GridPos** &p) const

  Gets the cell at the specified grid position.

- **GridCell** * **cell** (const **LatLng** &p)

  Gets the cell that covers the specified location.

- void **cells** (const **Polygon** &p, std::list< **GridCell** * > &outCells)

  Returns a list containing the grid cells covered by the provided **Polygon**(p. 326).

- void **cells** (const **Polygon** &p, std::list< const **GridCell** * > &outCells) const

  Returns a list containing the grid cells covered by the provided **Polygon**(p. 326).

- void **cells** (const **Circle** &inC, std::list< **GridCell** * > &outCells)

  Returns a list containing the grid cells covered by the provided **Circle**(p. 86).

- void **cells** (const **Circle** &inC, std::list< const **GridCell** * > &outCells) const

  Returns a list containing the grid cells covered by the provided **Circle**(p. 86).

- void **setParameters** (const **Disease** &d, const **ModelParameters** &mp, double HDI, double unemployment)

  Sets some simulation parameters.

- void **updateParameters** (double HDI, double unemployment)

  Updates some simulation parameters.

- void **notifyAboutCamp** (**Camp** *c)

  Notifies the Grid about the creation of a new **Camp**(p. 74).

- const **ModelParameters** & **mp** () const

Accessor for the **ModelParameters**(p. 301).

- const **Disease** & **disease** () const

  Accessor for the **Disease**(p. 168).

- void **populate** (const std::vector< **City** * > &cities)

  Spreads the population in 'cities' to the cells in the grid based on the city's distribution with its area as cutoff.

- void **init** (const std::vector< **Region** * > &regions)

  Initializes the Grid.

- void **initializeGrid** (const std::vector< **PVArea** * > &v)
- void **step** (const std::vector< **Region** * > &regions)

  Advances the Grid one timestep.

- void **expose** (const **Action** &inA)

  Expose the Grid to an **Action**(p. 19).

- void **setCombatGrid** (const **CombatGrid** *cg)

  Mutator for the **CombatGrid**(p. 101).

- **GridCell** * **getCellForNearestCamp** (**LatLng** p, double &dist)

  Gets the cell in which the camp that is closest to the provided point is located.

Private Member Functions

- void **setCellNeighbors** ()

  Sets the neighbors of all cells.

Private Attributes

- int **mFactions**

  Number of ethnic groups.

- double **mHDI**

  The HDI parameter.

- double **mUnemployment**

  The HDI parameter.

- double **mRegionFoodSurplus**
- double **mRegionFoodDeficit**
- double **mTotalPopulation**
- double * **mTotalInitialPopulation**

  Initial population of each group in the entire grid.

- double ** **mInitialPopulation**

An array that contains the initial population of each ethnic group in each cell where mInitial-Population[i][j] means the population of faction j in cell i - i refering to the active array.

- double * **mResettlers**
- **GridCell** ** **mCell**

  Array of pointers to active cells - size = mActive.

- **GridCell** ** **mCellP**

  Array of pointers to cells - size = mCells.

- std::vector< **Camp** * > **mCamps**

  A vector containing all camps.

- int * **mVindex**

  Indices of cells when clustering.

- double * **mVx**

  X-coordinates of cell centers when clustering.

- double * **mVy**

  Y-coordinates of cell centers when clustering.

- double * **mVw**

  Weights of cells when clustering.

- const **ModelParameters** * **mModelParameters**

  An object containing various model parameters.

- const **Disease** * **mDisease**

  An object containing disease parameters.

- const **CombatGrid** * **mCG**

  Pointer to the combat grid.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Grid** &g)

  For debugging purposes.

4.64.1 Detailed Description

This class represents the simulation grid.

Author:
Per Alexius

Date **Date**
2007/01/28 17:07:48

4.64.2 Constructor & Destructor Documentation

4.64.2.1 Grid::Grid (const Map & *amap*, double *cellSizeMeters*, int *factions*)

Creates a Grid for the provided **Map**(p. 292) with the specified cell size.

Parameters:

amap The map to create the Grid for.

cellSizeMeters The side of the cells in meters.

factions The number of factions excluding the all faction.

4.64.3 Member Function Documentation

4.64.3.1 Camp* Grid::camp (unsigned int *i*) const [inline]

Accessor for camps.

Parameters:

i The index in the mCamps vector.

Returns:

The specified camp or null if no such camp exists..

4.64.3.2 int Grid::camps () const [inline]

Accessor for the number of camps.

Returns:

The number of camps.

4.64.3.3 GridCell * Grid::cell (const LatLng & *p*)

Gets the cell that covers the specified location.

Returns:

The cell that covers the specified location or null if no such cell exists.

4.64.3.4 const GridCell* Grid::cell (const GridPos & *p*) const [inline]

Gets the cell at the specified grid position.

Returns:

The cell at the specified grid position or null if no such cell exists.

4.64.3.5 const GridCell* Grid::cell (int *r*, int *c*) const [inline]

Gets the cell with the specified row and column number.

Returns:
The cell with the specified row and column number or null if no such cell exists.

4.64.3.6 GridCell* Grid::cell (const GridPos & *p*) [inline]

Gets the cell at the specified grid position.

Returns:
The cell at the specified grid position or null if no such cell exists.

4.64.3.7 GridCell* Grid::cell (int *r*, int *c*) [inline]

Gets the cell with the specified row and column number.

Returns:
The cell with the specified row and column number or null if no such cell exists.

4.64.3.8 GridCell* Grid::cell (int *ind*) const [inline]

Gets the cell with the specified index in the active cells array.

Returns:
The cell with the specified index in the active cells array or null if no such cell exists

4.64.3.9 const double* const Grid::cellCenterCoordsX () const [inline]

Accessor for the cell center x coordinate array used for clustering.

Returns:
The cell center x coordinate array.

4.64.3.10 const double* const Grid::cellCenterCoordsY () const [inline]

Accessor for the cell center y coordinate array used for clustering.

Returns:
The cell center y coordinate array.

4.64.3.11 void Grid::cells (const Circle & *inC*, std::list< const GridCell * > & *outCells*) const

Returns a list containing the grid cells covered by the provided **Circle**(p. 86).

Parameters:

inC The **Circle**(p. 86) to get the grid cells for.

outCells A list that on return contains the grid cells covered by the provided **Circle**(p. 86).

4.64.3.12 void Grid::cells (const Circle & *inC*, std::list< GridCell * > & *outCells*)

Returns a list containing the grid cells covered by the provided **Circle**(p. 86).

Parameters:

inC The **Circle**(p. 86) to get the grid cells for.

outCells A list that on return contains the grid cells covered by the provided **Circle**(p. 86).

4.64.3.13 void Grid::cells (const Polygon & *p*, std::list< const GridCell * > & *outCells*) const

Returns a list containing the grid cells covered by the provided **Polygon**(p. 326).

Parameters:

p The **Polygon**(p. 326) to get the grid cells for.

outCells A list that on return contains the grid cells covered by the provided **Polygon**(p. 326).

4.64.3.14 void Grid::cells (const Polygon & *p*, std::list< GridCell * > & *outCells*)

Returns a list containing the grid cells covered by the provided **Polygon**(p. 326).

Parameters:

p The **Polygon**(p. 326) to get the grid cells for.

outCells A list that on return contains the grid cells covered by the provided **Polygon**(p. 326).

4.64.3.15 const CombatGrid* Grid::cg () const [inline]

Accessor for the cell center y coordinate array used for clustering.

Returns:

The cell center y coordinate array.

4.64.3.16 const Disease& Grid::disease () const [inline]

Accessor for the **Disease**(p. 168).

Returns:
The **Disease**(p. 168).

4.64.3.17 void Grid::expose (const Action & *inA*)

Expose the Grid to an **Action**(p. 19).

Parameters:
inA The **Action**(p. 19) to expose the Grid to.

4.64.3.18 int Grid::factions () const [inline]

Accessor for the number of factions, excluding the 'all' faction.

Returns:
The number of factions, excluding the 'all' faction.

4.64.3.19 GridCell * Grid::getCellForNearestCamp (LatLng *p*, double & *dist*)

Gets the cell in which the camp that is closest to the provided point is located.

Parameters:
p The point to measure from.
dist Contains the distance in meters to the nearest **Camp**(p. 74) on successful return, undefined otherwise.

Returns:
The cell in which the nearest camp is located or null if there are no camps.

4.64.3.20 double Grid::HDI () const [inline]

Accessor for the HDI parameter.

Returns:
The HDI parameter.

4.64.3.21 double Grid::initialPopulation (int *i*, int *f* = 0) const [inline]

Accessor for the initial population of a faction in a cell.

Parameters:
i The cell index (in the active array).
f The faction index.

Returns:
The initial population for the specified cell and faction.

4.64.3.22 const ModelParameters& Grid::mp () const [inline]

Accessor for the **ModelParameters**(p. 301).

Returns:
The **ModelParameters**(p. 301).

4.64.3.23 void Grid::notifyAboutCamp (Camp * *c*)

Notifies the Grid about the creation of a new **Camp**(p. 74).

Parameters:
c The newly created **Camp**(p. 74).

4.64.3.24 void Grid::populate (const std::vector< City * > & *cities*)

Spreads the population in 'cities' to the cells in the grid based on the city's distribution with its area as cutoff.

Parameters:
cities A vector of Cities to spread the population from.

4.64.3.25 void Grid::setCombatGrid (const CombatGrid * *cg*) [inline]

Mutator for the **CombatGrid**(p. 101).

Parameters:
cg The **CombatGrid**(p. 101).

4.64.3.26 void Grid::setParameters (const Disease & *d*, const ModelParameters & *mp*, double *HDI*, double *unemployment*)

Sets some simulation parameters.

Parameters:
d The disease from the scenario object.
mp The model parameters from the simulation object.
HDI The HDI parameter.
unemployment The unemployment parameter.

4.64.3.27 double Grid::totalInitialPopulation (int *f* = 0) const [inline]

Accessor for the initial population of a faction in the entire grid.

Parameters:
f The faction index.

Returns:
The initial population for the specified faction.

4.64.3.28 double Grid::totalPopulation () const [inline]

Accessor for the total population in the entire grid.

Returns:
The total population.

4.64.3.29 double Grid::unemployment () const [inline]

Accessor for the unemployment parameter.

Returns:
The unemployment parameter.

4.64.3.30 void Grid::updateParameters (double *HDI*, double *unemployment*)

Updates some simulation parameters.

Parameters:
HDI The HDI parameter.
unemployment The unemployment parameter.

4.64.4 Friends And Related Function Documentation

4.64.4.1 std::ostream& operator<< (std::ostream & *o*, const Grid & *g*) [friend]

For debugging purposes.

Parameters:
o The stream to write to.
g The Grid to print.

Returns:
The provided ostream with the Grid written to it.

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

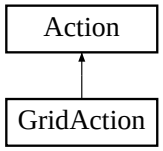
- Grid.h
- Grid.cpp

4.65 GridAction Class Reference

This class represents an **Action**(p. 19) that affects the grid.

`#include <Action.h>`

Inheritance diagram for GridAction::



Public Member Functions

- **GridAction** (**Grid** &e, const **Shape** &area, const **Element** *performer, const std::vector< **GridEffect** > &effects, double fraction)

  Constructor.

- const **Shape** & **location** () const

  Gets the area.

- const **Element** * **performer** () const

  Gets the performer of this action.

- int **effects** () const

  Gets the number of effects of this action.

- **GridEffect effect** (int index) const

  Gets ths specified effect.

Private Attributes

- const **Shape** & **mLocation**

  The location.

- const **Element** * **mPerformer**

  The performer.

- const std::vector< **GridEffect** > & **mEffects**

  A vector of effects for this **Action**(p. 19).

- double **mFraction**

  The fraction of full effect this action should have.

4.65.1 Detailed Description

This class represents an **Action**(p. 19) that affects the grid.

Author:
Per Alexius

Date:
2006/03/06 12:55:07

4.65.2 Constructor & Destructor Documentation

4.65.2.1 GridAction::GridAction (Grid & *e*, const Shape & *area*, const Element * *performer*, const std::vector< GridEffect > & *effects*, double *fraction*) [inline]

Constructor.

Parameters:
e The target of this **Action**(p. 19).
area The area to expose
performer The performer of the **Action**(p. 19).
effects A vector of effects for this **Action**(p. 19).
fraction The fraction of full effect this action should have.

4.65.3 Member Function Documentation

4.65.3.1 GridEffect GridAction::effect (int *index*) const [inline]

Gets ths specified effect.

Parameters:
index The index of the effect to fetch.

Returns:
The specified effect.

4.65.3.2 int GridAction::effects () const [inline]

Gets the number of effects of this action.

Returns:
The number of effects of this action.

4.65.3.3 const Shape& GridAction::location () const [inline]

Gets the area.

Returns:
The area

4.65.3.4 const Element* GridAction::performer () const [inline]

Gets the performer of this action.

Returns:
The performer or null if there is none.

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

- Action.h

4.66 GridCell Class Reference

This class represents a cell in the **Grid**(p. 227).

`#include <GridCell.h>`

Public Member Functions

- **GridCell** (**Grid** &g, int activeIndex, const double *corners, int nGrp)
  Creates a GridCell.
- ~**GridCell** ()
  Destructor.
- int **index** () const
  Accessor for the index.
- int **pos** () const
  Accessor for the position.
- int **row** () const
  Accessor for the row.
- int **col** () const
  Accessor for the column.
- **LatLng center** () const
  Accessor for the center coordinate.
- double **squDistanceTo** (const **GridCell** &g) const
  Gets the square of the distance in meters between this cell and the provided cell.
- **GridCell** * **neighbor** (int i)
  Accessor for the neighbors.
- void **setNeighbor** (**GridCell** *cell, eNeighbor dir)
  Mutator for the neighbors.
- void **makeCalculatedTSCurrentTS** ()
  Swaps the attribute buffers.
- void **addOverlappingRegion** (const **Region** &r)
- int **numOverlappingRegions** () const
- void **init** ()
  Initializes the attributes for this cell.
- void **update** ()
  Updates the attributes for this cell, i.e. calculates the next timestep.
- void **expose** (eAllPV pv, const **EthnicFaction** &faction, double size)

Updates the derived attributes for this cell.

- void **adjustValues** ()

  Checks so that the limits of displaced, sheltered, protected etc doesn't exceed the population number and recalculates sums and averages. Implemented in order to handle **PVRegion**(p. 371) *initialization.*

- void **handleRoundOffErrors** ()

  Checks for round off errors.

- int **dailyShots** () const

  Not applicable since there is no fighting.

- double **smoothedShots** () const

  Not applicable since there is no fighting.

- double **weight** () const
- double **pvfGet** (ePVF pv, int f=0) const
- void **pvfSet** (ePVF pv, int f, double value)
- void **pvfSetR** (ePVF pv, int f, double value)
- void **pvfAdd** (ePVF pv, int f, double value)
- void **pvfAddR** (ePVF pv, int f, double value)
- double **pvGet** (ePV pv) const
- void **pvSet** (ePV pv, double value)
- void **pvSetR** (ePV pv, double value)
- void **pvAdd** (ePV pv, double value)
- void **pvAddR** (ePV pv, double value)
- double **pdfGet** (eDerivedF pv, int f=0) const
- void **pdfSet** (eDerivedF pv, int f, double value)
- void **pdfAdd** (eDerivedF pv, int f, double value)
- void **pdfReset** ()
- double **pdGet** (eDerived pv) const
- void **pdSet** (eDerived pv, double value)
- void **pdAdd** (eDerived pv, double value)
- void **pdReset** ()
- double **pcfGet** (ePreCalcF pv, int f=0) const
- void **pcfSet** (ePreCalcF pv, int f, double value)
- void **pcfAdd** (ePreCalcF pv, int f, double value)
- void **pcfReset** ()
- double **pcGet** (ePreCalc pv) const
- void **pcSet** (ePreCalc pv, double value)
- void **pcAdd** (ePreCalc pv, double value)
- void **pcReset** ()
- void **pvAllSet** (eAllPV pv, int f, double value)

  Sets a PV:s value (write index) from the index in the eAllPV enumeration.

- void **pvAllSetR** (eAllPV pv, int f, double value)

  Sets a PV:s value (read index) from the index in the eAllPV enumeration.

- void **recalculateAllFaction** ()

Calculates the sum or mean for the all faction for all non derived pv:s with factions.

- void **setSumR** (ePVF pv)

  Calculates the sum over the factions for a PV and stores it as the value of the all faction.

- void **setPopulationWeightedAverageR** (ePVF pv)

  Calculates the population weighted average over the factions for a PV and stores it as the value of the all faction.

- void **doDerived** ()
- void **doPrecalculated** ()
- double **popDensity** () const
- double **bestFoodStorage** () const
- double **bestWaterCapacity** () const
- double **expectedTension** () const
- std::ostream & **print2** (std::ostream &o)
- std::vector< const **Region** * > **regions** () const

Static Public Member Functions

- void **setNumberOfFactions** (int num)

  Mutator for the faction count.

- double **areaKm2** ()
- void **handleRoundOffErrorsPositive** (double *data, int size=1)
- void **handleRoundOffErrorsPercent** (double *data, int size=1)
- void **handleRoundOffErrorsFraction** (double *data, int size=1)

Private Member Functions

- void **position** (const double *corner)

  Sets the coordinates for the corner points of this cell.

- double **pvrGet** (eRegionPV pvr)
- double **regionParam** (eRegionParameter param)
- double **regionPVFGet** (ePVF pvf, int f=0)
- void **doPopulation** (double *data)
- void **doDisplaced** (double *data)
- void **doSheltered** (double *data)
- void **doViolence** (double *data)
- void **doPerceivedThreat** (double *data)
- void **doInsurgents** (double *data)
- void **doEthnicTension** (double *data)
- void **doFractionCrimeVictims** (double *data)
- void **doHousingUnits** (double *data)
- void **doStoredFood** (double *data)
- void **doFoodConsumption** (double *data)
- void **doFarmStoredFood** (double *data)
- void **doMarketedFood** (double *data)
- void **doFoodDays** (double *data)

- void **doWaterConsumption** (double *data)
- void **doWaterDays** (double *data)
- void **doSusceptible** (double *data)
- void **doInfected** (double *data)
- void **doRecovered** (double *data)
- void **doDeadDueToDisease** (double *data)
- void **doFractionInfected** (double *data)
- void **doFractionRecovered** (double *data)
- void **doFractionNoWork** (double *data)
- void **doProtected** (double *data)
- void **doFractionNoMedical** (double *data)
- void **doFractionNoFood** (double *data)
- void **doSuppliedWater** (double *data)
- void **doFractionNoWater** (double *data)
- void **doInfrastructure** (double *data)
- void **doTEST** (double *data)
- void **doDDisaffection** (double *data)
- void **doDPolarization** (double *data)
- void **doDHoused** (double *data)
- void **doDAvailableFood** (double *data)
- void **doDFoodDeprivation** (double *data)
- void **doDWaterSurplusDeficit** (double *data)
- void **doDWaterDeprivation** (double *data)
- void **doNothingF** (ePVF pv)
- void **doNothing** (ePV pv)
- void **exposePercent** (double *data, const **EthnicFaction** &fac, double size)
- void **exposeDisplaced** (double *data, const **EthnicFaction** &fac, double size)
- void **exposeSheltered** (double *data, const **EthnicFaction** &fac, double size)
- void **exposeProtected** (double *data, const **EthnicFaction** &fac, double size)
- void **exposeInsurgents** (double *data, const **EthnicFaction** &fac, double size)
- void **exposeFraction** (double *data, double size)
- void **exposeFoodDays** (double *data, double size)
- void **exposeWaterDays** (double *data, double size)
- void **exposeFractionInfected** (double *data, double size)
- void **exposeFractionRecovered** (double *data, double size)

Private Attributes

- **Grid** & **mGrid**

  Reference(p. 378) *to the grid this cell is a part of.*

- int **mIndex**

  This cell's index in the active cells array.

- int **mPos**

  *This cell's position in the grid i.e. r * nCol + c.*

- int **mRow**

  The row of this cell.

- int **mCol**

  The column of this cell.

- **GridCell** ** **mNeighbor**

  An array of pointers to this cells neighboring cells.

- **LatLng mCenter**

  The center coordinate of this cell.

- std::vector< const **Region** * > **mRegions**

  Contains all regions overlapping this cell.

- int **mReadInd**
- int **mWriteInd**
- double ** **mPVF** [2]
- double * **mPV** [2]
- double * **mPreCalcF**
- double * **mPreCalc**
- double * **mDerivedF**
- double * **mDerived**
- double **mWellWaterFraction**

Static Private Attributes

- int **sFactions** = 1
- double **sCellAreaKm2** = 0

Friends

- std::ostream & **operator**<< (std::ostream &o, const **GridCell** &c)

  For debugging purposes.

4.66.1 Detailed Description

This class represents a cell in the **Grid**(p. 227).

Author:
Per Alexius

Date **Date**
2007/01/28 17:07:49

4.66.2 Constructor & Destructor Documentation

4.66.2.1 GridCell::GridCell (Grid & *g*, int *activeIndex*, const double * *corners*, int *nGrp*)

Creates a GridCell.

Parameters:

g The **Grid**(p. 227) to which this cell belongs.

activeIndex The index of this cell in the active array.

corners The corners of this cell in lat lng.

nGrp The number of factions excluding the all faction.

4.66.3 Member Function Documentation

4.66.3.1 LatLng GridCell::center () const [inline]

Accessor for the center coordinate.

Returns:

The center coordinate.

4.66.3.2 int GridCell::col () const [inline]

Accessor for the column.

Returns:

The column.

4.66.3.3 int GridCell::dailyShots () const [inline]

Not applicable since there is no fighting.

Returns:

0.

4.66.3.4 void GridCell::expose (eAllPV *pv*, const EthnicFaction & *faction*, double *size*)

Updates the derived attributes for this cell.

Parameters:

attr The attribute to expose.

faction The faction to expose.

size The magnitude of the effect.

4.66.3.5 int GridCell::index () const [inline]

Accessor for the index.

Returns:

The index.

4.66.3.6 GridCell* GridCell::neighbor (int *i*) [inline]

Accessor for the neighbors.

Parameters:
i The neighbor as defined in eNeighbor.

Returns:
The neighbor in the specified direction or null if no such neighbor exists.

4.66.3.7 int GridCell::pos () const [inline]

Accessor for the position.

Returns:
The position.

4.66.3.8 void GridCell::position (const double * *corner*) [inline, private]

Sets the coordinates for the corner points of this cell.

Parameters:
corner An array of size 8 containing the four corner points of this cell (x0 y0 x1 y1...). The corners are ordered clockwise from the bottom left corner.

4.66.3.9 void GridCell::pvAllSet (eAllPV *pv*, int *f*, double *value*)

Sets a PV:s value (write index) from the index in the eAllPV enumeration.

Parameters:
pv The index in the eAllPV enumeration.

f The faction index.

value The value.

4.66.3.10 void GridCell::pvAllSetR (eAllPV *pv*, int *f*, double *value*)

Sets a PV:s value (read index) from the index in the eAllPV enumeration.

Parameters:
pv The index in the eAllPV enumeration.

f The faction index.

value The value.

4.66.3.11 int GridCell::row () const [inline]

Accessor for the row.

Returns:
The row.

4.66.3.12 void GridCell::setNeighbor (GridCell * *cell*, eNeighbor *dir*) [inline]

Mutator for the neighbors.

Parameters:
cell The neighboring cell.

dir The direction this neighbor are located in.

4.66.3.13 void GridCell::setNumberOfFactions (int *num*) [inline, static]

Mutator for the faction count.

Parameters:
num The number of factions.

4.66.3.14 void GridCell::setPopulationWeightedAverageR (ePVF *pv*)

Calculates the population weighted average over the factions for a PV and stores it as the value of the all faction.

Parameters:
pv The index in the ePVF enumeration.

4.66.3.15 void GridCell::setSumR (ePVF *pv*)

Calculates the sum over the factions for a PV and stores it as the value of the all faction.

Parameters:
pv The index in the ePVF enumeration.

4.66.3.16 double GridCell::smoothedShots () const [inline]

Not applicable since there is no fighting.

Returns:
0.

4.66.3.17 double GridCell::squDistanceTo (const GridCell & *g*) const [inline]

Gets the square of the distance in meters between this cell and the provided cell.

Parameters:

g The cell to measure the distance to.

Returns:

The square of the distance in meters between this cell and the provided cell.

4.66.4 Friends And Related Function Documentation

4.66.4.1 std::ostream& operator<< (std::ostream & *o*, const GridCell & *c*) [friend]

For debugging purposes.

Parameters:

o The stream to write to.

c The cell to print.

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

- GridCell.h
- GridCell.cpp
- GridCellPV.cpp

4.67 GridDataHandler Class Reference

Helper object that provides an interface for accessing data from the grid based on layer name, i.e. the name of the process variable.

`#include <GridDataHandler.h>`

Public Member Functions

- **GridDataHandler** (**Grid** &grid, **CombatGrid** &cg, const std::vector< **Faction** * > &facVec)

  Creates a GridDataHandler for the provided grid and combat grid.

- ~**GridDataHandler** ()

  Destructor.

- const **Grid** & **grid** () const

  Accessor for the **Grid**(p. 227).

- const **CombatGrid** & **combatGrid** () const

  Accessor for the **CombatGrid**(p. 101).

- const std::string & **stanceLayerName** (int i) const

  Maps the stance layer index to the layer name.

- int **stanceLayers** () const

  Returns the number of stance layers.

- double **ps** (int cellIndex, int stanceIndex) const

  Gets the value of a stance variable for the specified cell.

- void **layer** (const std::string &lay, const **Reference** &fac, int size, int32_t *index, double *&outData)

  Fetches process variable values for the specified cells, process variable and faction.

- void **extractGridData** ()

  Copies data from the simulation **Grid**(p. 227) *to this object so that it will be accessible for clients.*

Private Member Functions

- void **layer** (int lay, int fac, int size, int32_t *index, double *&outLayer)

  Fetches process variable values for the specified cells, process variable and faction.

Private Attributes

- std::map< std::string, int > **mLayerNameToIndex**

  Maps layer name to its index in the mGridData.

- const std::vector< **Faction** * > & **mFactions**
  Reference(p. 378) *to the Scenario's faction vector.*
- double ** **mGridData**
  Array for the values of all process variables in all cells.
- **Grid** & **mGrid**
  Reference(p. 378) *to the* **Grid**(p. 227) *to handle data from.*
- **CombatGrid** & **mCG**
  Reference(p. 378) *to the* **CombatGrid**(p. 101) *to handle data from.*
- int **mNumActive**
  Number of active cells.
- int **mNumLayers**
  Number of layers.
- int **mStanceStartIndex**
  The index in the mGridData[][] for the first stance layer.
- std::vector< std::string > **mStanceLayerName**

4.67.1 Detailed Description

Helper object that provides an interface for accessing data from the grid based on layer name, i.e. the name of the process variable.

Author:
Per Alexius

Date:
2006/10/10 09:36:51

4.67.2 Constructor & Destructor Documentation

4.67.2.1 GridDataHandler::GridDataHandler (Grid & *grid*, CombatGrid & *cg*, const std::vector< Faction * > & *facVec*)

Creates a GridDataHandler for the provided grid and combat grid.

Parameters:
grid The **Grid**(p. 227).
cg The **CombatGrid**(p. 101).

4.67.3 Member Function Documentation

4.67.3.1 const CombatGrid& GridDataHandler::combatGrid () const [inline]

Accessor for the **CombatGrid**(p. 101).

Returns:
A **Reference**(p. 378) to the **CombatGrid**(p. 101).

4.67.3.2 void GridDataHandler::extractGridData ()

Copies data from the simulation **Grid**(p. 227) to this object so that it will be accessible for clients.

Called by the **Engine**(p. 189) via **Buffer**(p. 67) when simulation data should be transfered from the simulation to the **Buffer**(p. 67), for example after each timestep and after initialization.

4.67.3.3 const Grid& GridDataHandler::grid () const [inline]

Accessor for the **Grid**(p. 227).

Returns:
A **Reference**(p. 378) to the **Grid**(p. 227).

4.67.3.4 void GridDataHandler::layer (const std::string & *lay*, const Reference & *fac*, int *size*, int32_t * *index*, double *& *outData*)

Fetches process variable values for the specified cells, process variable and faction.

Parameters:
lay The name of the process variable.

fac A **Reference**(p. 378) to the faction.

size The number of cells to fetch values for.

index An array of size elements containing the indices in the active cells array of the cells for which to fetch the values.

outData An array of size elements that on return will contain the values for the specified cells.

4.67.3.5 void GridDataHandler::layer (int *lay*, int *fac*, int *size*, int32_t * *index*, double *& *outData*) [private]

Fetches process variable values for the specified cells, process variable and faction.

Parameters:
lay The process variable number according to the eAttribute enumeration.

fac The faction index.

size The number of cells to fetch values for.

index An array of size elements containing the indices in the active cells array of the cells for which to fetch the values.

outData An array of size elements that on return will contain the values for the specified cells.

4.67.3.6 double GridDataHandler::ps (int *cellIndex*, int *stanceIndex*) const [inline]

Gets the value of a stance variable for the specified cell.

Parameters:

cellIndex The index (in the active array) of the cell.

stanceIndex The index among the stance layers.

Returns:

The value of a stance variable for the specified cell.

4.67.3.7 const std::string& GridDataHandler::stanceLayerName (int *i*) const [inline]

Maps the stance layer index to the layer name.

Parameters:

i The index among the stance layers

Returns:

The name of the layer.

4.67.3.8 int GridDataHandler::stanceLayers () const [inline]

Returns the number of stance layers.

Returns:

The number of stance layers.

4.67.4 Member Data Documentation

4.67.4.1 double** GridDataHandler::mGridData [private]

Array for the values of all process variables in all cells.

The structure is as follows:

mGridData[number of active cells][number of layers]

where cells are ordered from top left to bottom right.

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

- GridDataHandler.h
- GridDataHandler.cpp

4.68 GridEffect Class Reference

GridEffect represents an effect on a process variable.

`#include <GridEffect.h>`

Public Member Functions

- **GridEffect** (eAllPV pv, double s, **EthnicFaction** *f)
  Constructor.

Public Attributes

- eAllPV **mPV**
  The attribute the effect effects.
- double **mSeverity**
  The severity of the effect - a value between [-10, 10].
- **EthnicFaction** * **mFaction**
  Pointer to the **EthnicFaction**(p. 209) *the effect effects.*

4.68.1 Detailed Description

GridEffect represents an effect on a process variable.

GridEffects are used in Activities that effect the **Grid**(p. 227) directly.

4.68.2 Constructor & Destructor Documentation

4.68.2.1 GridEffect::GridEffect (eAllPV *pv*, double *s*, EthnicFaction * *f*) `[inline]`

Constructor.

Parameters:

pv The attribute number as defined in eAllPV.

s The severity of the effect.

f A pointer to the **Faction**(p. 212) the effect referse to.

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

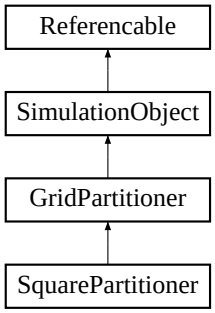
- GridEffect.h

4.69 GridPartitioner Class Reference

An abstract base class for all GridPartitioners.

`#include <GridPartitioner.h>`

Inheritance diagram for GridPartitioner::



Public Member Functions

- **GridPartitioner** (const **DataObject** &d)

  Creates a GridPartitioner from the provided **DataObject**(p. 145).

- virtual ~**GridPartitioner** ()

  Destructor.

- virtual **Grid** * **createGrid** (const **Map** &m, int numEthnicFactions) const =0

  Creates a **Grid**(p. 227).

4.69.1 Detailed Description

An abstract base class for all GridPartitioners.

Author:
Per Alexius

Date:
2006/03/06 14:23:07

4.69.2 Constructor & Destructor Documentation

4.69.2.1 GridPartitioner::GridPartitioner (const DataObject & *d*) [inline]

Creates a GridPartitioner from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.69.3 Member Function Documentation

4.69.3.1 virtual Grid* GridPartitioner::createGrid (const Map & *m*, int *numEthnicFactions*) const [pure virtual]

Creates a **Grid**(p. 227).

Parameters:

m The map to lay the **Grid**(p. 227) over.

numEthnicFactions The number of ethnic factions.

Returns:

The newly created **Grid**(p. 227).

Implemented in **SquarePartitioner** (p. 455).

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

- GridPartitioner.h

4.70 GridPos Class Reference

A GridPos represents a position in the **Grid**(p. 227).

`#include <GridPos.h>`

Public Member Functions

- **GridPos** ()

  Default constructor.

- **GridPos** (int ir, int ic)

  Constructs a GridPos (ir, ic).

- bool **operator==** (const **GridPos** &p) const

  Compares two GridPos.

- bool **operator<** (const **GridPos** &p) const

  The order of GridPos is as follows. The top left (0, 0) is the smallest, the top second left (0, 1) is the second smallest etc.

Public Attributes

- int **c**

  The column.

- int **r**

  The row.

Friends

- std::ostream & **operator<<** (std::ostream &os, const **GridPos** &p)

  For debugging purposes.

4.70.1 Detailed Description

A GridPos represents a position in the **Grid**(p. 227).

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

- GridPos.h

4.71 hashReferenceP Struct Reference

Function object used to create a hashcode for a **Reference**(p. 378). Needed by hash_map.

`#include <Reference.h>`

Public Member Functions

- size_t **operator()** (const **Reference** *const key) const

  Produces a hashcode for the **Reference**(p. 378) *pointed to by key. Since there may onbly be one* **Reference**(p. 378) *object for each reference the address will do.*

4.71.1 Detailed Description

Function object used to create a hashcode for a **Reference**(p. 378). Needed by hash_map.

Author:
Per Alexius

Date:
2006/05/24 12:32:11

4.71.2 Member Function Documentation

4.71.2.1 size_t hashReferenceP::operator() (const Reference *const *key*) const [inline]

Produces a hashcode for the **Reference**(p. 378) pointed to by key. Since there may onbly be one **Reference**(p. 378) object for each reference the address will do.

Parameters:
key Pointer to the **Reference**(p. 378) for which to create a hashcode.

Returns:
A hashcode for the specified **Reference**(p. 378).

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

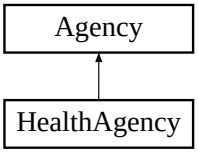
- Reference.h

4.72 HealthAgency Class Reference

Class containing functionality for controlling HealthAgencyTeams.

`#include <Agency.h>`

Inheritance diagram for HealthAgency::



Public Member Functions

- **HealthAgency** (const std::vector< **AgencyTeam** * > &teams, **Grid** &g)

  Constructor.

Private Member Functions

- bool **severeProblem** ()

  Determines if we have a severe resource problem. If we do - then the weights for the clustering algorithm is set.

4.72.1 Detailed Description

Class containing functionality for controlling HealthAgencyTeams.

Author:
Per Alexius

Date:
2006/10/02 16:01:25

4.72.2 Constructor & Destructor Documentation

4.72.2.1 HealthAgency::HealthAgency (const std::vector< AgencyTeam * > & *teams*, Grid & *g*) [inline]

Constructor.

Parameters:
teams A vector containing this Agency's teams.

g A reference to the **Grid**(p. 227).

4.72.3 Member Function Documentation

4.72.3.1 bool HealthAgency::severeProblem () [private, virtual]

Determines if we have a severe resource problem. If we do - then the weights for the clustering algorithm is set.

Returns:
True if we have a severe disease problem, false otherwise.

Implements **Agency** (p. 28).

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

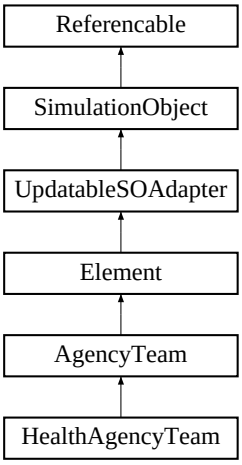
- Agency.h
- Agency.cpp

4.73 HealthAgencyTeam Class Reference

Class representing a HealthAgencyTeam.

`#include <AgencyTeam.h>`

Inheritance diagram for HealthAgencyTeam::



Public Member Functions

- **HealthAgencyTeam** (const **DataObject** &d)
  Constructor that creates an **AgencyTeam**(p. 32) *from the provided* **DataObject**(p. 145).
- double **calculateNeed** ()
  Calculates the need for medical support among the population in this team's area of influence.
- void **act** (**Time** now)
  Performs this team's actions.

4.73.1 Detailed Description

Class representing a HealthAgencyTeam.

Author:
Per Alexius

Date **Date**
2006/10/10 09:35:59

4.73.2 Constructor & Destructor Documentation

4.73.2.1 HealthAgencyTeam::HealthAgencyTeam (const DataObject & *d*) `[inline]`

Constructor that creates an **AgencyTeam**(p. 32) from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.73.3 Member Function Documentation

4.73.3.1 void HealthAgencyTeam::act (Time *now*) [virtual]

Performs this team's actions.

Reduces the proportion infected people by an average of 5% in all cells in this team's area of influence until the team's capacity limit is met.

Parameters:
now The current simulation time.

Implements **AgencyTeam** (p. 35).

4.73.3.2 double HealthAgencyTeam::calculateNeed () [virtual]

Calculates the need for medical support among the population in this team's area of influence.

Need for medical support is calculated as follows: For all cells in this team's area of influence with population p $>$ 0.5 persons - add f $*$ p (where f is the fraction of the population that is infected) to the total need.

Returns:
The need for medical support in this team's area of influence represented as the number of infected persons.

Reimplemented from **AgencyTeam** (p. 36).

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

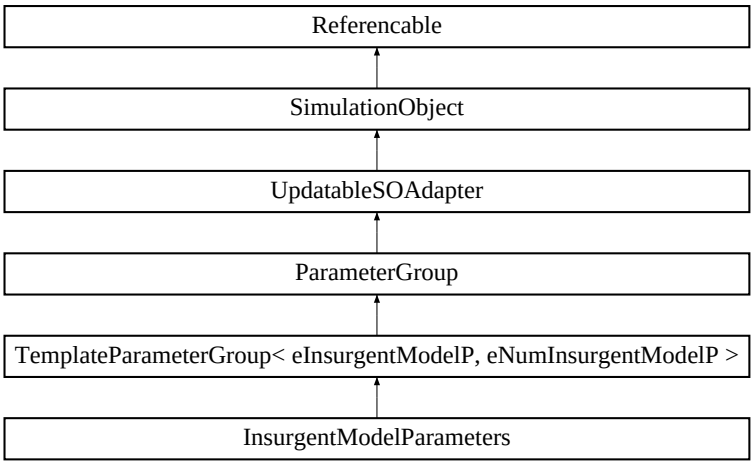
- AgencyTeam.h
- AgencyTeam.cpp

4.74 InsurgentModelParameters Class Reference

The insurgent model parameter group. This refers to the implemented insurgent model but is not used yet since the **ModelParameters**(p. 301) class is still used. It has been left here for future use.

`#include <ValidParameterGroups.h>`

Inheritance diagram for InsurgentModelParameters::



Public Member Functions

- **InsurgentModelParameters** (const **DataObject** &d)

4.74.1 Detailed Description

The insurgent model parameter group. This refers to the implemented insurgent model but is not used yet since the **ModelParameters**(p. 301) class is still used. It has been left here for future use.

Author:
Per Alexius

Date **Date**
2007/01/24 13:13:25

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

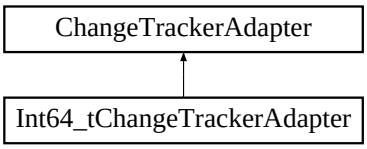
- ValidParameterGroups.h

4.75 Int64_tChangeTrackerAdapter Class Reference

The Int64_tChangeTrackerAdapter keeps track of changes in **StratmasInt64_t**(p. 468) objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for Int64_tChangeTrackerAdapter::



Public Member Functions

- **Int64_tChangeTrackerAdapter** (**StratmasInt64_t** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 264) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **StratmasInt64_t** & **mObject**

  The adapted **DataObject**(p. 145).

- int64_t **mLast**

  The last value written.

4.75.1 Detailed Description

The Int64_tChangeTrackerAdapter keeps track of changes in **StratmasInt64_t**(p. 468) objects.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.75.2 Constructor & Destructor Documentation

4.75.2.1 Int64_tChangeTrackerAdapter::Int64_tChangeTrackerAdapter (StratmasInt64_t & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.75.3 Member Function Documentation

4.75.3.1 bool Int64_tChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 264) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 264) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.75.3.2 ostream & Int64_tChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.76 IOHandler Class Reference

Class providing helpers for file IO.

`#include <IOHandler.h>`

Static Public Member Functions

- void **enableFileOutput** ()

  Make calls to dumpToFile to actually dump to file. Currently this is set from **Environment**(p. 197) *iff the user specifies an output dir. Note that compiling in DEBUG is also activates file output.*

- void **dumpToFile** (const std::string &toDump, const std::string &filename, std::ios_base::openmode mode=std::ios_base::trunc)

  Dumps a string to the specified file, if we'er allowed to do so.

Static Private Attributes

- bool **sWriteToFile** = false

  True if we are allowed to write to files.

4.76.1 Detailed Description

Class providing helpers for file IO.

Author:
Per Alexius

Date:
2006/07/24 10:14:35

4.76.2 Member Function Documentation

4.76.2.1 void IOHandler::dumpToFile (const std::string & *toDump*, const std::string & *filename*, std::ios_base::openmode *mode* = std::ios_base::trunc) [static]

Dumps a string to the specified file, if we'er allowed to do so.

Parameters:

toDump The string to dump.

filename The name of the file (in the output directory) to dump to.

mode The openmode of the file.

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

- IOHandler.h
- IOHandler.cpp

4.77 IPAddress Class Reference

Class representing an IP address.

`#include <IPAddress.h>`

Public Member Functions

- **IPAddress** (const std::string &ip)
  Creates an IPAddress from a string.
- **IPAddress** (const **IPAddress** &ip)
  Copy constructor.
- std::string **toString** () const
  Produces a string representation of this IPAddress.
- bool **operator==** (const **IPAddress** &ip) const
  Equality operator.
- bool **operator==** (const std::string &ip) const
  Equality operator for strings.

Private Attributes

- int **mParts** [4]

Friends

- std::ostream & **operator<<** (std::ostream &o, const **IPAddress** &ip)
  For printing to ostreams.

4.77.1 Detailed Description

Class representing an IP address.

Author:
Per Alexius

Date:
2006/05/23 09:57:34

4.77.2 Constructor & Destructor Documentation

4.77.2.1 IPAddress::IPAddress (const std::string & *ip*)

Creates an IPAddress from a string.

Ignores leading whitespace and all characters after the last integer part.

Parameters:
ip The string to create the IPAddress from.

4.77.2.2 IPAddress::IPAddress (const IPAddress & *ip*)

Copy constructor.

Parameters:
ip The IPAddress to copy.

4.77.3 Member Function Documentation

4.77.3.1 bool IPAddress::operator== (const std::string & *ip*) const

Equality operator for strings.

Parameters:
ip A string representation of the other IPAddress.

Returns:
True if the IPAddresses are equal.

4.77.3.2 bool IPAddress::operator== (const IPAddress & *ip*) const

Equality operator.

Parameters:
ip The other IPAddress.

Returns:
True if the IPAddresses are equal.

4.77.3.3 string IPAddress::toString () const

Produces a string representation of this IPAddress.

Returns:
A string representation of this IPAddress.

4.77.4 Friends And Related Function Documentation

4.77.4.1 std::ostream& operator<< (std::ostream & *o*, const IPAddress & *ip*) [friend]

For printing to ostreams.

Parameters:
o The stream to write to.

ip The IPAddress to write.

Returns:
The provided stream with the IPAddress written to it.

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

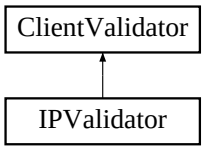
- IPAddress.h
- IPAddress.cpp

4.78 IPValidator Class Reference

Class that stores ip numbers that the server should allow connections from.

`#include <IPValidator.h>`

Inheritance diagram for IPValidator::



Public Member Functions

- bool **getValidIPsFromFile** (const std::string &filename)
  Gets valid ip numbers from the specified file.
- bool **addValidIP** (const std::string &ipToAdd)
  Adds a valid ip number.
- bool **isValidIP** (const std::string &ipToValidate)
  Checks if an ip number is valid.
- virtual bool **isValidClient** (const **Socket** *socket)
- int **numValidIPs** () const

Static Private Member Functions

- bool **isStringIP** (const std::string &str)
  Finds out if the provided string may be interpreted as an IP address.

Private Attributes

- std::set< std::string > **mIPSet**
  A set with ip numbers that the server should allow connections from.

4.78.1 Detailed Description

Class that stores ip numbers that the server should allow connections from.

Author:
Per Alexius

Date:
2006/07/21 13:35:29

4.78.2 Member Function Documentation

4.78.2.1 bool IPValidator::addValidIP (const std::string & *ipToAdd*)

Adds a valid ip number.

Parameters:
ipToAdd The ip number to add.

4.78.2.2 bool IPValidator::getValidIPsFromFile (const std::string & *filename*)

Gets valid ip numbers from the specified file.

Parameters:
filename The name of the file.

Returns:
True if the file was read successfully, false otherwise.

4.78.2.3 bool IPValidator::isStringIP (const std::string & *str*) [static, private]

Finds out if the provided string may be interpreted as an IP address.

Parameters:
The string to check.

Returns:
True if the string could be interpreted as an IP address, false otherwise.

4.78.2.4 bool IPValidator::isValidIP (const std::string & *ipToValidate*)

Checks if an ip number is valid.

Parameters:
ipToValidate The ip number to validate.

Returns:
True if the ip numner is valid, false otherwise.

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

- IPValidator.h
- IPValidator.cpp

4.79 LatLng Class Reference

The LatLng class represents a geografic location indicated by degrees latitude and longitude.

`#include <LatLng.h>`

Public Member Functions

- **LatLng** ()
  Constructs a point representing nowhere.
- **LatLng** (double lat, double lng)
  Constructs a point.
- virtual ~**LatLng** ()
  Destructor.
- bool **nowhere** () const
  Checks if this point is nowhere.
- void **setPos** (double lat, double lng)
  Sets the position of this LatLng.
- double **lat** () const
  Accessor for this point's latitude.
- double **lng** () const
  Accessor for this point's longitude.
- double **squDistanceTo** (const **LatLng** &p) const
  Returns the square of the distance between this point and the point p.
- **ProjCoord toCoord** () const
  Returns the projection of this point using the current **Projection**(p. 348).
- bool **operator==** (const **LatLng** &p) const
  Equality operator.
- bool **operator!=** (const **LatLng** &p) const
  Not-equal-to operator.

Protected Attributes

- double **mLat**
  The latitude of this point.
- double **mLng**
  The longitude of this point.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **LatLng** &p)

  For debugging purposes.

4.79.1 Detailed Description

The LatLng class represents a geografic location indicated by degrees latitude and longitude.

Author:
Per Alexius

Date:
2005/06/13 11:19:06

4.79.2 Constructor & Destructor Documentation

4.79.2.1 LatLng::LatLng (double *lat*, double *lng*) [inline]

Constructs a point.

Parameters:
lat The latitude.
lng The longitude.

4.79.3 Member Function Documentation

4.79.3.1 double LatLng::lat () const [inline]

Accessor for this point's latitude.

Returns:
This point's latitude.

4.79.3.2 double LatLng::lng () const [inline]

Accessor for this point's longitude.

Returns:
This point's longitude.

4.79.3.3 bool LatLng::nowhere () const [inline]

Checks if this point is nowhere.

Returns:
True if this point is nowhere, false otherwise.

4.79.3.4 bool LatLng::operator!= (const LatLng & *p*) const [inline]

Not-equal-to operator.

Parameters:
p The point to compare with.

Returns:
True if the points are not equal, false otherwise.

4.79.3.5 bool LatLng::operator== (const LatLng & *p*) const [inline]

Equality operator.

Parameters:
p The point to compare with.

Returns:
True if the points are equal, false otherwise.

4.79.3.6 void LatLng::setPos (double *lat*, double *lng*) [inline]

Sets the position of this LatLng.

Parameters:
lat The latitude.
lng The longitude.

4.79.3.7 double LatLng::squDistanceTo (const LatLng & *p*) const [inline]

Returns the square of the distance between this point and the point p.

Parameters:
p The point to measure the distance to.

Returns:
The square of the distance between this point and the point p in meters.

4.79.3.8 ProjCoord LatLng::toCoord () const [inline]

Returns the projection of this point using the current **Projection**(p. 348).

Returns:
The projection of this point using the current **Projection**(p. 348).

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

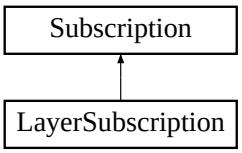
- LatLng.h

4.80 LayerSubscription Class Reference

LayerSubscription represents a subscription for one grid layer, e.g one process variable for all active cells.

`#include <Subscription.h>`

Inheritance diagram for LayerSubscription::



Public Member Functions

- **LayerSubscription** (DOMElement *n, **Buffer** &buf, bool sbe)

  Creates a subscription from a DOMElement, e.g. an xml representation.

- ~**LayerSubscription** ()

  Destructor.

- void **getSubscribedData** (std::ostream &o)

  Writes an XML representation of the subscribed data to the provided stream.

Private Attributes

- std::string **mLayer**

  The name of the process variable.

- const **Reference** * **mFaction**

  The faction this **Subscription**(p. 500) *refers to.*

- unsigned int **mLength**

  The number of cells of interest.

- int32_t * **mIndex**

  The indices (among active cells) of the cells of interest.

- bool **mSessionBigEndian**

  Keeps track of if we have to swap byte order.

4.80.1 Detailed Description

LayerSubscription represents a subscription for one grid layer, e.g one process variable for all active cells.

Due to performance considerations the layer is represented as a Base64 encoded array of doubles where the first element in the array is the value for the top left active cell, the second is the value for the top second left cell etc. down to the bottom right active cell. mLayer is the name of the layer and mFaction is the **Reference**(p. 378) that identifies which faction the subscription refers to.

Author:
Per Alexius

Date:
2006/07/05 14:49:47

4.80.2 Constructor & Destructor Documentation

4.80.2.1 LayerSubscription::LayerSubscription (DOMElement * *n*, Buffer & *buf*, bool *sbe*)

Creates a subscription from a DOMElement, e.g. an xml representation.

Parameters:
n The DOMElement from which this subscription should be created.
buf The **Buffer**(p. 67) from which data should be fetched.
sbe True if the client that submitted this subscription runs on a big endian plattform.

4.80.3 Member Function Documentation

4.80.3.1 void LayerSubscription::getSubscribedData (std::ostream & *o*) [virtual]

Writes an XML representation of the subscribed data to the provided stream.

Parameters:
o The stream to write to.

Implements **Subscription** (p. 501).

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

- Subscription.h
- Subscription.cpp

4.81 lessActivityPointer Struct Reference

Function object for less-than operator for pointer to Activities.

`#include <Activity.h>`

Public Member Functions

- bool **operator()** (const **Activity** *const a1, const **Activity** *const a2) const

  Less-than operator for pointers to Activities.

4.81.1 Detailed Description

Function object for less-than operator for pointer to Activities.

An **Activity**(p. 21) is less than another **Activity**(p. 21) if its start time is earlier than the other Activitiy's.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.81.2 Member Function Documentation

4.81.2.1 bool lessActivityPointer::operator() (const Activity *const *a1*, const Activity *const *a2*) const [inline]

Less-than operator for pointers to Activities.

Parameters:

a1 The first **Activity**(p. 21).

a2 The second **Activity**(p. 21).

Returns:
True if the first **Activity**(p. 21) is less than the other **Activity**(p. 21), false otherwise.

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

- Activity.h

4.82 lessGridCellPtr Struct Reference

Function object for less-than operator for pointer to GridCells.

`#include <GridCell.h>`

Public Member Functions

- bool **operator()** (const **GridCell** *c1, const **GridCell** *c2)

  Less-than operator for pointers to GridCells.

4.82.1 Detailed Description

Function object for less-than operator for pointer to GridCells.

A cell is less than another cell if it has a smaller row number - or if the row numbers are equal - has a smaller column number.

Author:
Per Alexius

Date:
2007/01/28 17:07:49

4.82.2 Member Function Documentation

4.82.2.1 bool lessGridCellPtr::operator() (const GridCell * *c1*, const GridCell * *c2*) [inline]

Less-than operator for pointers to GridCells.

Parameters:
c1 The first cell.

c2 The second cell.

Returns:
True if the first cell is less than the other cell, false otherwise.

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

- GridCell.h

4.83 lessPresenceObjectPointer Struct Reference

Function object for less-than operator for pointer to PresenceObjects.

`#include <PresenceObject.h>`

Public Member Functions

- bool **operator()** (const **PresenceObject** *const p1, const **PresenceObject** *const p2) const

  Less-than operator for pointers to PresenceObjects.

4.83.1 Detailed Description

Function object for less-than operator for pointer to PresenceObjects.

A **PresenceObject**(p. 338) is less than another **PresenceObject**(p. 338) if it has a lower cell index.

Author:
Per Alexius

Date:
2006/03/06 12:55:11

4.83.2 Member Function Documentation

4.83.2.1 bool lessPresenceObjectPointer::operator() (const PresenceObject *const *p1*, const PresenceObject *const *p2*) const [inline]

Less-than operator for pointers to PresenceObjects.

Parameters:
p1 The first **PresenceObject**(p. 338).
p2 The second **PresenceObject**(p. 338).

Returns:
True if the first **PresenceObject**(p. 338) is less than the other **PresenceObject**(p. 338), false otherwise.

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

- PresenceObject.h

4.84 lessReferenceP Struct Reference

Function object used to compare const **Reference**(p. 378) pointers. Needed by std::map.

`#include <Reference.h>`

Public Member Functions

- bool **operator()** (const **Reference** *const r1, const **Reference** *const r2) const
  Compares the References pointed to by r1 and r2.

4.84.1 Detailed Description

Function object used to compare const **Reference**(p. 378) pointers. Needed by std::map.

Author:
Per Alexius

Date:
2006/05/24 12:32:11

4.84.2 Member Function Documentation

4.84.2.1 bool lessReferenceP::operator() (const Reference *const *r1*, const Reference *const *r2*) const [inline]

Compares the References pointed to by r1 and r2.

Parameters:
r1 A pointer to a **Reference**(p. 378).
r2 A pointer to a **Reference**(p. 378).

Returns:
true if the **Reference**(p. 378) pointed to by r1 is less than the **Reference**(p. 378) pointed to by r2.

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

- Reference.h

4.85 lessTypeP Struct Reference

Function object used to compare const **Type**(p. 528) pointers. Needed by std::map.

`#include <Type.h>`

Public Member Functions

- bool **operator()** (const **Type** *const t1, const **Type** *const t2) const
  Compares the Types pointed to by t1 and t2.

4.85.1 Detailed Description

Function object used to compare const **Type**(p. 528) pointers. Needed by std::map.

Author:
Per Alexius

Date:
2006/03/02 17:06:56

4.85.2 Member Function Documentation

4.85.2.1 bool lessTypeP::operator() (const Type *const *t1*, const Type *const *t2*) const [inline]

Compares the Types pointed to by t1 and t2.

Parameters:
t1 A pointer to a **Type**(p. 528).
t2 A pointer to a **Type**(p. 528).

Returns:
true if the **Type**(p. 528) pointed to by t1 is less than the **Type**(p. 528) pointed to by t2.

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

- Type.h

4.86 Line Class Reference

Class representing a line. Used when parsing Polygons.

`#include <XMLHelper.h>`

Public Member Functions

- **Line** (const DOMElement &n)
  Creates a Line from the provided DOMElement.
- const std::string & **identifier** () const
  Accessor for the identifier.
- const **Point** & **p1** () const
  Accessor for the start point.
- const **Point** & **p2** () const
  Accessor for the end point.

Private Attributes

- std::string **mId**
  The id of the line.
- **Point mP1**
  The start point.
- **Point mP2**
  The end point.

4.86.1 Detailed Description

Class representing a line. Used when parsing Polygons.

Author:
Per Alexius

Date:
2007/01/24 13:13:27

4.86.2 Constructor & Destructor Documentation

4.86.2.1 Line::Line (const DOMElement & *n*) `[inline]`

Creates a Line from the provided DOMElement.

Parameters:
n The DOMElement to create this Line from.

4.86.3 Member Function Documentation

4.86.3.1 const std::string& Line::identifier () const [inline]

Accessor for the identifier.

Returns:
The identifier.

4.86.3.2 const Point& Line::p1 () const [inline]

Accessor for the start point.

Returns:
The start point.

4.86.3.3 const Point& Line::p2 () const [inline]

Accessor for the end point.

Returns:
The end point.

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

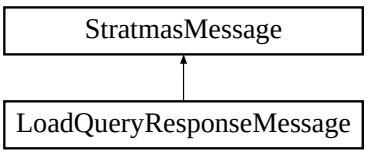
- XMLHelper.h
- XMLHelper.cpp

4.87 LoadQueryResponseMessage Class Reference

Class representing the LoadQueryResponseMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for LoadQueryResponseMessage::



Public Member Functions

- **LoadQueryResponseMessage** (const **Server** &s)

  Creates a LoadQueryResponseMessage that fetches its information from the specified **Server**(p. 403).

- void **toXML** (std::ostream &o) const

  Produces the XML representation of this message.

Private Attributes

- const **Server** & **mServer**

  The **Server**(p. 403) *from which information should be fetched.*

4.87.1 Detailed Description

Class representing the LoadQueryResponseMessage.

Author:
Per Alexius

Date **Date**
2006/03/06 14:23:12

4.87.2 Constructor & Destructor Documentation

4.87.2.1 LoadQueryResponseMessage::LoadQueryResponseMessage (const Server & *s*) `[inline]`

Creates a LoadQueryResponseMessage that fetches its information from the specified **Server**(p. 403).

Parameters:
s The **Server**(p. 403) to fetch information from.

4.87.3 Member Function Documentation

4.87.3.1 void LoadQueryResponseMessage::toXML (std::ostream & *o*) const [virtual]

Produces the XML representation of this message.

Parameters:

o The stream to which the message is written

Implements **StratmasMessage** (p. 473).

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

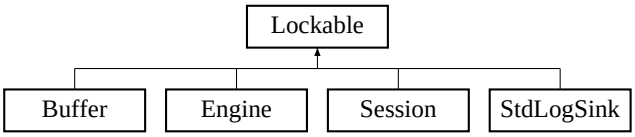
- StratmasMessage.h
- StratmasMessage.cpp

4.88 Lockable Class Reference

Wrapper around a mutex.

`#include <Lockable.h>`

Inheritance diagram for Lockable::



Public Member Functions

- boost::mutex & **mutex** () const

Private Attributes

- boost::mutex **mMutex**

  The mutex.

4.88.1 Detailed Description

Wrapper around a mutex.

Author:
Per Alexius

Date:
2006/07/05 07:42:41

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

- Lockable.h

4.89 LogEnd Class Reference

Placeholder class used to mark the end of a log message.

`#include <LogStream.h>`

4.89.1 Detailed Description

Placeholder class used to mark the end of a log message.

Author:
Daniel Ahlin

Date:
2006/07/25 14:52:00

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

- LogStream.h

4.90 LogMessage Class Reference

This class represents a log message.

`#include <LogStream.h>`

Public Member Functions

- ~**LogMessage** ()
  Destroys the LogMessage, posting it if not posted.
- **LogMessage** (**LogStream** *logStream)
  Creates a new log message.
- template<class T> **LogMessage** & **operator**<< (T t)
  Writes the provided object to this Message.
- const std::string **getMessage** () const
- **LogMessage** & **operator**<< (const **LogEnd** &end)
  Terminate the message by writing an instance of **LogEnd**(p. 286) *to it.*

Private Attributes

- std::ostringstream **mMessage**
  A message describing the error.
- **LogStream** * **mLogStream**

4.90.1 Detailed Description

This class represents a log message.

Author:
Daniel Ahlin

Date:
2006/07/25 14:52:00

4.90.2 Constructor & Destructor Documentation

4.90.2.1 LogMessage::LogMessage (LogStream * *logStream*) `[inline]`

Creates a new log message.

Parameters:
t The object to write.

Returns:
A reference to this **Error**(p. 205).

4.90.3 Member Function Documentation

4.90.3.1 LogMessage & LogMessage::operator<< (const LogEnd & *end*)

Terminate the message by writing an instance of **LogEnd**(p. 286) to it.

Parameters:
t The object to write.

Returns:
A reference to this **Error**(p. 205).

4.90.3.2 template<class T> LogMessage& LogMessage::operator<< (T *t*) [inline]

Writes the provided object to this Message.

Parameters:
t The object to write.

Returns:
A reference to this message.

4.90.4 Member Data Documentation

4.90.4.1 LogStream* LogMessage::mLogStream [private]

The log this message was created to be logged in. By being set to null, logStream also serves to flag that the message is already sent and that no more appending to this message is allowed.

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

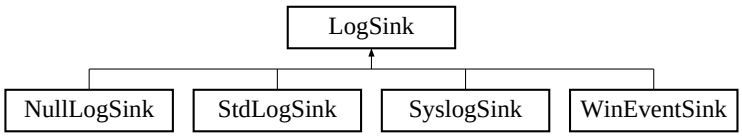
- LogStream.h
- LogStream.cpp

4.91 LogSink Class Reference

This class represents capabilities of a log sink.

`#include <LogStream.h>`

Inheritance diagram for LogSink::



Public Member Functions

- virtual void **sink** (const **LogMessage** *const message)=0

  Posts the provided message to the log stream.

4.91.1 Detailed Description

This class represents capabilities of a log sink.

Author:
Daniel Ahlin

Date **Date**
2006/07/25 14:52:00

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

- LogStream.h

4.92 LogStream Class Reference

This class is serves as a logging facility.

`#include <LogStream.h>`

Public Member Functions

- virtual ~**LogStream** ()

  Destroys the LogStream and releases any used **LogSink**(p. 289).

- **LogStream** ()

  Creates a new LogStream that logs to cerr.

- **LogStream** (**LogSink** *logSink)

  Creates a new LogStream that uses the provided **LogSink**(p. 289). *The LogStream will adopt the* **LogSink**(p. 289) *and delete it on destruction or change of logSink.*

- virtual void **postMessage** (**LogMessage** *message)

  Commits a message and then releases it.

- virtual void **setLogSink** (**LogSink** *newSink)

  Switches to a new sink. Note that the provided sink should be new'ed and that LogStream will adopt it (and delete it on destruction or change of **LogSink**(p. 289)*).*

- template<class T> **LogMessage** & **operator**<< (T t)

  Creates a new message and writes the provided object to it.

Private Attributes

- **LogSink** * **mLogSink**

4.92.1 Detailed Description

This class is serves as a logging facility.

Author:
Daniel Ahlin

Date:
2006/07/25 14:52:00

4.92.2 Member Function Documentation

4.92.2.1 template<class T> LogMessage& LogStream::operator<< (T *t*) [inline]

Creates a new message and writes the provided object to it.

Parameters:
t The object to write.

Returns:
A reference to this message.

4.92.2.2 virtual void LogStream::postMessage (LogMessage * *message*) [inline, virtual]

Commits a message and then releases it.

Parameters:
message The object to commit.

4.92.2.3 virtual void LogStream::setLogSink (LogSink * *newSink*) [inline, virtual]

Switches to a new sink. Note that the provided sink should be new'ed and that LogStream will adopt it (and delete it on destruction or change of **LogSink**(p. 289)).

TODO/NOTE It is questionable if this should be done without synchronization (depends on the atomicity of =).

Parameters:
message The object to commit.

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

- LogStream.h
- LogStream.cpp

4.93 Map Class Reference

Class representing the map the simulation concerns.

`#include <Map.h>`

Public Member Functions

- **Map** (const **Shape** &s)

  Creates a Map.

- ~**Map** ()

  Destructor.

- **Shape** & **borders** () const

  Access the longitude of the center coordinate.

- double **cenLng** () const

  Access the longitude of the center coordinate.

- double **cenLat** () const

  Access the latitude of the center coordinate.

- double **minX** () const

  Access the minimum x-value for the Map.

- double **maxX** () const

  Access the maximum x-value for the Map.

- double **minY** () const

  Access the minimum y-value for the Map.

- double **maxY** () const

  Access the maximum y-value for the Map.

- double **width** () const

  Access the width of the Map.

- double **height** () const

  Access the height of the Map.

- const **Projection** & **proj** () const

  Access the current **Projection**(p. 348).

- const **Shape** * **getRegionForPoint** (const **ProjCoord** &p) const

  Finds out in which subshape the specified point is located.

Private Attributes

- **Shape** * **mBorders**

  The **Shape**(p. 412) *constituting the map.*

- double **mCenLat**

  Latitude of the center of the Map's bounding box.

- double **mCenLng**

  Longitude of the center of the Map's bounding box.

- double **mMinX**

  Leftmost coordinate of the Map.

- double **mMaxX**

  Rightmost coordinate of the Map.

- double **mMinY**

  Minimum y-coordinate of the Map.

- double **mMaxY**

  Maximum y-coordinate of the Map.

- **Projection** * **mProj**

  Pointer to the projection used for this Map.

4.93.1 Detailed Description

Class representing the map the simulation concerns.

Author:
Per Alexius

Date:
2006/02/28 17:48:19

4.93.2 Constructor & Destructor Documentation

4.93.2.1 Map::Map (const Shape & *s*)

Creates a Map.

Parameters:
s The **Shape**(p. 412) for this map.

4.93.3 Member Function Documentation

4.93.3.1 Shape& Map::borders () const [inline]

Access the longitude of the center coordinate.

Returns:
Longitude of the center coordinate

4.93.3.2 double Map::cenLat () const [inline]

Access the latitude of the center coordinate.

Returns:
Latitude of the center coordinate

4.93.3.3 double Map::cenLng () const [inline]

Access the longitude of the center coordinate.

Returns:
Longitude of the center coordinate

4.93.3.4 const Shape * Map::getRegionForPoint (const ProjCoord & *p*) const

Finds out in which subshape the specified point is located.

Parameters:
p The point.

Returns:
The first found subschape that contains the point, or null if no such subshape could be found.

4.93.3.5 double Map::height () const [inline]

Access the height of the Map.

Returns:
Height of the Map

4.93.3.6 double Map::maxX () const [inline]

Access the maximum x-value for the Map.

Returns:
Maximum x-value for the Map

4.93.3.7 double Map::maxY () const [inline]

Access the maximum y-value for the Map.

Returns:
Maximum y-value for the Map

4.93.3.8 double Map::minX () const [inline]

Access the minimum x-value for the Map.

Returns:
Minimum x-value for the Map

4.93.3.9 double Map::minY () const [inline]

Access the minimum y-value for the Map.

Returns:
Minimum y-value for the Map

4.93.3.10 const Projection& Map::proj () const [inline]

Access the current **Projection**(p. 348).

Returns:
Current **Projection**(p. 348)

4.93.3.11 double Map::width () const [inline]

Access the width of the Map.

Returns:
Width of the Map

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

- Map.h
- Map.cpp

4.94 Mapper Class Reference

This class is used to map References to their corresponding **DataObject**(p. 145).

`#include <Mapper.h>`

Static Public Member Functions

- void **reg** (const **DataObject** *c)

  Registers the provided **DataObject**(p. 145) *with this Mapper.*

- void **dereg** (const **DataObject** &d)

  Deregisters the provided **DataObject**(p. 145) *from this mapper.*

- **DataObject** * **map** (const **Reference** &ref)

  Maps the provided **Reference**(p. 378) *to its corresponding* **DataObject**(p. 145).

- void **clear** ()

  Erases all mappings.

Static Private Attributes

- MapType **mMap**

  Contains the mappings between **Reference**(p. 378) *and* **DataObject**(p. 145).

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Mapper** &m)

  For debugging purposes.

4.94.1 Detailed Description

This class is used to map References to their corresponding **DataObject**(p. 145).

Author:
Per Alexius

Date **Date**
2006/07/03 14:18:23

4.94.2 Member Function Documentation

4.94.2.1 void Mapper::dereg (const DataObject & *d*) [inline, static]

Deregisters the provided **DataObject**(p. 145) from this mapper.

Parameters:
d The **DataObject**(p. 145) to deregister.

4.94.2.2 DataObject* Mapper::map (const Reference & *ref*) [inline, static]

Maps the provided **Reference**(p. 378) to its corresponding **DataObject**(p. 145).

Parameters:
ref The **Reference**(p. 378) to find a **DataObject**(p. 145) for.

Returns:
The **DataObject**(p. 145) for the provided **Reference**(p. 378) of null if no such **DataObject**(p. 145) was found..

4.94.2.3 void Mapper::reg (const DataObject * *c*) [inline, static]

Registers the provided **DataObject**(p. 145) with this Mapper.

Parameters:
c The **DataObject**(p. 145) to register.

4.94.3 Friends And Related Function Documentation

4.94.3.1 std::ostream& operator<< (std::ostream & *o*, const Mapper & *m*) [friend]

For debugging purposes.

Parameters:
o The stream to write to.

m The Mapper to print.

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

- Mapper.h
- Buffer.cpp

4.95 MemEntityResolver Class Reference

This class provides schemas to the xml parser used in **XMLHandler**(p. 582).

`#include <MemEntityResolver.h>`

Public Member Functions

- **MemEntityResolver** (XMLEntityResolver *xmlEntityResolverFallback=0, EntityResolver *entityResolverFallback=0)

  Creates a new MemEntitiyResolver that tries to deliver entitys from an internal lookup table, if unable to fullfill the request it will consult provided fallback resolver, if any.

- virtual ~**MemEntityResolver** ()

  Destroys this resolver.

- virtual InputSource * **resolveEntity** (const XMLCh *const publicId, const XMLCh *const systemId)
- virtual InputSource * **resolveEntity** (XMLResourceIdentifier *resourceIdentifier)

  Returns an InputSource for the provided resource idententifier. If unable to fullfill the request and a fallback were provided in the constructor, the function will return the answer of the fallback.

- virtual InputSource * **resolve** (const std::string &publicId)

  Returns an InputSource for the provided resource idententifier, or null if unable to fullfill the request.

Private Member Functions

- virtual InputSource * **resolve** (const XMLCh *const publicId)

  Returns an InputSource for the provided resource idententifier, or null if unable to fullfill the request.

Private Attributes

- XMLEntityResolver * **mpXMLEntityResolverFallback**
- EntityResolver * **mpEntityResolverFallback**

4.95.1 Detailed Description

This class provides schemas to the xml parser used in **XMLHandler**(p. 582).

Author:
Daniel Ahlin

Date:
2006/07/21 13:35:29

4.95.2 Member Function Documentation

4.95.2.1 InputSource * MemEntityResolver::resolve (const std::string & *systemId*) [virtual]

Returns an InputSource for the provided resource idententifier, or null if unable to fullfill the request.

Returns:
an InputSource for the resourceidentifier, or null if unable to provide one. The returned InputSource is owned by the caller which is responsible to clean up the memory.

4.95.2.2 InputSource * MemEntityResolver::resolve (const XMLCh *const *systemId*) [private, virtual]

Returns an InputSource for the provided resource idententifier, or null if unable to fullfill the request.

Returns:
an InputSource for the resourceidentifier, or null if unable to provide one. The returned InputSource is owned by the caller which is responsible to clean up the memory.

4.95.2.3 InputSource * MemEntityResolver::resolveEntity (XMLResourceIdentifier * *resourceIdentifier*) [virtual]

Returns an InputSource for the provided resource idententifier. If unable to fullfill the request and a fallback were provided in the constructor, the function will return the answer of the fallback.

Returns:
An input source for the entity, or null if unable to provide one, the returned InputSource is owned by the parser which is responsible to clean up the memory.

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

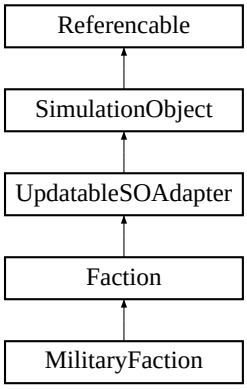
- MemEntityResolver.h
- MemEntityResolver.cpp

4.96 MilitaryFaction Class Reference

The MilitaryFaction class contains the Stratmas server representation of a MilitaryFaction.

`#include <Faction.h>`

Inheritance diagram for MilitaryFaction::



Public Member Functions

- **MilitaryFaction** (const **DataObject** &d)

4.96.1 Detailed Description

The MilitaryFaction class contains the Stratmas server representation of a MilitaryFaction.

Author:
Per Alexius

Date:
2006/07/05 14:49:43

4.96.2 Constructor & Destructor Documentation

4.96.2.1 MilitaryFaction::MilitaryFaction (const DataObject & *d*) [inline]

Constructor that creates a MilitaryFaction from a **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

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

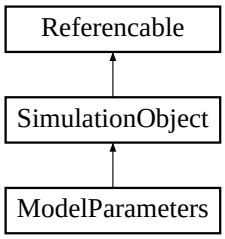
- Faction.h

4.97 ModelParameters Class Reference

The **SimulationObject**(p. 429) that corresponds to the ModelParameters type in the Stratmas xml schemas.

`#include <ModelParameters.h>`

Inheritance diagram for ModelParameters::



Public Member Functions

- **ModelParameters** (const **DataObject** &d)

  Creates a ModelParameters object from the provided **DataObject**(p. 145).

- ~**ModelParameters** ()

  Destructor.

- void **setDefault** ()

  Sets values to default.

- void **update** (const **Update** &u)

  Updates this object.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- double **mp** (eModelParameter param) const

Static Public Member Functions

- const char * **paramName** (eModelParameter param)

Private Member Functions

- void **getDataFromDataObject** (const **DataObject** &d)

Private Attributes

- double * **mParam**

Static Private Attributes

- std::map< std::string, eModelParameter > **sNameToIndex**

4.97.1 Detailed Description

The **SimulationObject**(p. 429) that corresponds to the ModelParameters type in the Stratmas xml schemas.

Author:
Per Alexius

Date:
2006/09/04 14:34:42

4.97.2 Constructor & Destructor Documentation

4.97.2.1 ModelParameters::ModelParameters (const DataObject & *d*)

Creates a ModelParameters object from the provided **DataObject**(p. 145).

Parameters:
d The data object to create this **TimeStepper**(p. 522) from.

4.97.3 Member Function Documentation

4.97.3.1 void ModelParameters::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.97.3.2 void ModelParameters::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

4.97.3.3 void ModelParameters::update (const Update & *u*) [virtual]

Updates this object.

Parameters:

u The **Update**(p. 567) to update this object with.

Implements **SimulationObject** (p. 431).

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

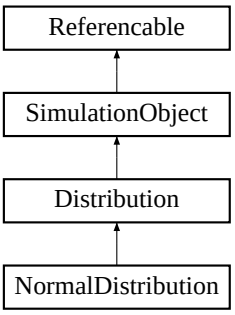
- ModelParameters.h
- ModelParameters.cpp

4.98 NormalDistribution Class Reference

Normal distribution.

`#include <Distribution.h>`

Inheritance diagram for NormalDistribution::



Public Member Functions

- **NormalDistribution** (double sigma)

  Constructor.

- **NormalDistribution** (const **DataObject** &d)

  Creates a NormalDistribution from the specified **DataObject**(p. 145).

- virtual ~**NormalDistribution** ()

  Destructor.

- double **f** (double x) const

  Gets the value of the distribution at distance x.

- void **update** (const **Update** &u)

  Updates this object.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Private Attributes

- double **mSigma**

  The diffusion measure.

- double **mK1**

  Distribution(p. 173) *constant.*

- double **mK2**
  Distribution(p. 173) *constant.*

4.98.1 Detailed Description

Normal distribution.

Author:
Per Alexius

Date:
2006/04/21 15:54:49

4.98.2 Constructor & Destructor Documentation

4.98.2.1 NormalDistribution::NormalDistribution (double *sigma*)

Constructor.

Parameters:
sigma Diffusion measure.

4.98.2.2 NormalDistribution::NormalDistribution (const DataObject & *d*)

Creates a NormalDistribution from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.98.3 Member Function Documentation

4.98.3.1 void NormalDistribution::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.98.3.2 double NormalDistribution::f (double *x*) const [inline, virtual]

Gets the value of the distribution at distance x.

Parameters:
x The distance.

Returns:
The value of the distribution at distance x.

Implements **Distribution** (p. 175).

4.98.3.3 void NormalDistribution::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Distribution** (p. 175).

4.98.3.4 void NormalDistribution::update (const Update & *u*) [virtual]

Updates this object.

Parameters:

u The **Update**(p. 567) to update this object with.

Reimplemented from **Distribution** (p. 175).

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

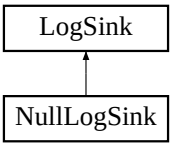
- Distribution.h
- Distribution.cpp

4.99 NullLogSink Class Reference

This class implements **LogSink**(p. 289), suppresing output.

`#include <LogStream.h>`

Inheritance diagram for NullLogSink::



Public Member Functions

- virtual void **sink** (const **LogMessage** *const message) const

  Posts the provided message to the log stream.

4.99.1 Detailed Description

This class implements **LogSink**(p. 289), suppresing output.

Author:
Daniel Ahlin

Date **Date**
2006/07/25 14:52:00

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

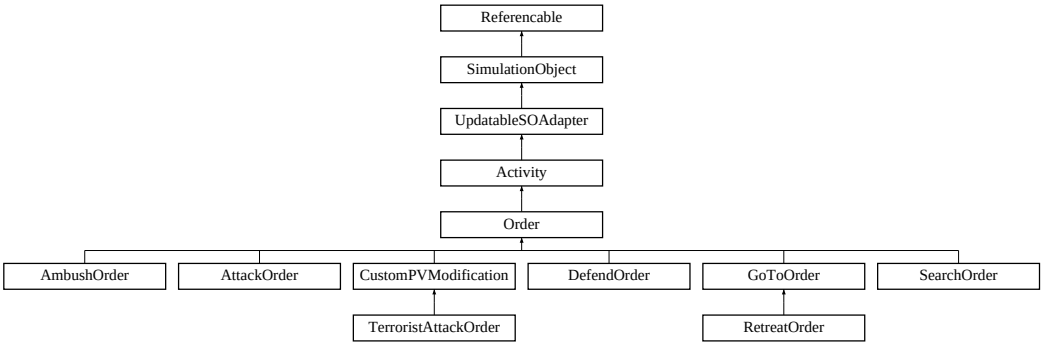
- LogStream.h

4.100 Order Class Reference

Abstract super class for all orders.

`#include <Activity.h>`

Inheritance diagram for Order::



Public Member Functions

- **Order** ()

  Default constructor.

- **Order** (const **DataObject** &d)

  Creates an Order from the provided **DataObject**(p. 145).

- virtual ~**Order** ()

  Destructor.

- virtual bool **isActive** (**Time** t)

  Checks if this activity is active at time t.

- **Shape** * **location** () const

  Accessor for the area.

- virtual bool **isCarriedOut** ()

  Checks if this order is carried out.

- virtual void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- virtual void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- virtual void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- virtual void **replaceObject** (**DataObject** &newObject, int64_t initiator)

  Replaces the **SimulationObject**(p. 429) *with the same reference as the provided* **DataObject**(p. 145) *with a new* **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145).

- virtual void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- virtual void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- virtual double **combatFactor** () const =0

  Accessor for the combat factor.

Protected Attributes

- bool **mCarriedOut**

  Indicates if this order has been executed.

- **Shape** * **mLocation**

  The location of this order (nullable).

4.100.1 Detailed Description

Abstract super class for all orders.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.100.2 Constructor & Destructor Documentation

4.100.2.1 Order::Order (const DataObject & *d*)

Creates an Order from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.100.3 Member Function Documentation

4.100.3.1 void Order::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:

toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 564).

Reimplemented in **CustomPVModification** (p. 141).

4.100.3.2 virtual double Order::combatFactor () const [pure virtual]

Accessor for the combat factor.

Returns:

The combat factor.

Implemented in **CustomPVModification** (p. 141), **AttackOrder** (p. 56), **DefendOrder** (p. 166), **AmbushOrder** (p. 43), **GoToOrder** (p. 226), and **SearchOrder** (p. 402).

4.100.3.3 void Order::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:

b The **Buffer**(p. 67) to extract data to.

Reimplemented from **Activity** (p. 22).

Reimplemented in **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 514), **DefendOrder** (p. 166), and **AmbushOrder** (p. 43).

4.100.3.4 virtual bool Order::isActive (Time *t*) [inline, virtual]

Checks if this activity is active at time t.

Parameters:

t The time for which to check.

Returns:

True if this activity is active at the specified time.

Implements **Activity** (p. 23).

Reimplemented in **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 166), **AmbushOrder** (p. 43), and **SearchOrder** (p. 402).

4.100.3.5 virtual bool Order::isCarriedOut () [inline, virtual]

Checks if this order is carried out.

Returns:

True if this order is carried out, false otherwise.

4.100.3.6 Shape* Order::location () const [inline, virtual]

Accessor for the area.

Returns:
The area or null if this activity does not have an area.

Implements **Activity** (p. 23).

4.100.3.7 void Order::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **Activity** (p. 23).

Reimplemented in **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 167), and **AmbushOrder** (p. 43).

4.100.3.8 void Order::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:
toRemove The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

Reimplemented in **CustomPVModification** (p. 143).

4.100.3.9 void Order::replaceObject (DataObject & *newObject*, int64_t *initiator*) [virtual]

Replaces the **SimulationObject**(p. 429) with the same reference as the provided **DataObject**(p. 145) with a new **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145).

Parameters:
newObject The **DataObject**(p. 145) to create the replacing object from.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.100.3.10 void Order::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Activity** (p. 24).

Reimplemented in **CustomPVModification** (p. 143), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 167), and **AmbushOrder** (p. 44).

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

- Activity.h
- Activity.cpp

4.101 ParameterEntry Struct Reference

This struct defines a parameter entry to be used with the **TemplateParameterGroup**(p. 509) class.

`#include <ParameterGroup.h>`

Public Attributes

- int **index**
- const char * **name**
- const char * **type**
- double **defaultValue**

4.101.1 Detailed Description

This struct defines a parameter entry to be used with the **TemplateParameterGroup**(p. 509) class.

Author:
Per Alexius

Date:
2006/09/04 14:39:16

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

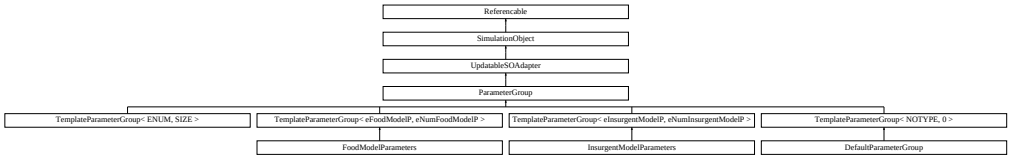
- ParameterGroup.h

4.102 ParameterGroup Class Reference

Abstract ParameterGroup.

`#include <ParameterGroup.h>`

Inheritance diagram for ParameterGroup::



Public Member Functions

- **ParameterGroup** (const **DataObject** &d)
- virtual void **prepareForSimulation** ()=0

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- std::ostream & **printMe** (std::ostream &o) const
- virtual std::ostream & **printMe** (std::ostream &o, std::string indent) const =0

4.102.1 Detailed Description

Abstract ParameterGroup.

Author:
Per Alexius

Date **Date**
2006/09/04 14:39:16

4.102.2 Member Function Documentation

4.102.2.1 virtual void ParameterGroup::prepareForSimulation () [pure virtual]

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Implemented in **TemplateParameterGroup< ENUM, SIZE >** (p. 512), **TemplateParameterGroup< eInsurgentModelP, eNumInsurgentModelP >** (p. 512), **TemplateParameterGroup< NOTYPE, 0 >** (p. 512), and **TemplateParameterGroup< eFoodModelP, eNumFoodModelP >** (p. 512).

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

- ParameterGroup.h

4.103 ParameterGroupEntry Struct Reference

This struct defines a parameter group entry to be used with the **TemplateParameterGroup**(p. 509) class.

`#include <ParameterGroup.h>`

Public Attributes

- const char * **name**

4.103.1 Detailed Description

This struct defines a parameter group entry to be used with the **TemplateParameterGroup**(p. 509) class.

Author:
Per Alexius

Date:
2006/09/04 14:39:16

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

- ParameterGroup.h

4.104 ParserErrorReporter Class Reference

Error(p. 205) reporter for the DOMParser.

`#include <ParserErrorReporter.h>`

Public Member Functions

- bool **errorsOccurred** () const
  Checks if any errors occurred.
- void **warning** (const SAXParseException &toCatch)
  Handles warning exceptions.
- void **error** (const SAXParseException &toCatch)
  Handles error exceptions.
- void **fatalError** (const SAXParseException &toCatch)
  Handles fatalError exceptions.
- std::vector< **Error** > **errors** () const
  Accessor for the vector containing the errors.
- void **resetErrors** ()
  Resets the mErrorsOccurred flag.

Private Attributes

- std::vector< **Error** > **mErrors**
  A vector containing the errors.

4.104.1 Detailed Description

Error(p. 205) reporter for the DOMParser.

Author:
Per Alexius

Date:
2005/12/08 11:26:52

4.104.2 Member Function Documentation

4.104.2.1 void ParserErrorReporter::error (const SAXParseException & *toCatch*) [inline]

Handles error exceptions.

Parameters:
toCatch The exception to handle.

4.104.2.2 std::vector<Error> ParserErrorReporter::errors () const [inline]

Accessor for the vector containing the errors.

Returns:
The vector containing the errors.

4.104.2.3 bool ParserErrorReporter::errorsOccurred () const [inline]

Checks if any errors occurred.

Returns:
True if any errors occurred, false otherwise.

4.104.2.4 void ParserErrorReporter::fatalError (const SAXParseException & *toCatch*) [inline]

Handles fatalError exceptions.

Parameters:
toCatch The exception to handle.

4.104.2.5 void ParserErrorReporter::warning (const SAXParseException & *toCatch*) [inline]

Handles warning exceptions.

Parameters:
toCatch The exception to handle.

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

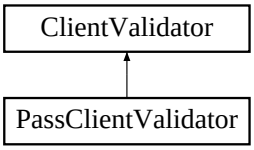
- ParserErrorReporter.h

4.105 PassClientValidator Class Reference

Class that allows any client to connect.

`#include <ClientValidator.h>`

Inheritance diagram for PassClientValidator::



Public Member Functions

- virtual bool **isValidClient** (const **Socket** *socket)

4.105.1 Detailed Description

Class that allows any client to connect.

Author:
Daniel Ahlin

Date:
2006/07/21 13:35:29

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

- ClientValidator.h

4.106 Point Class Reference

Class representing a point. Used when parsing Polygons.

`#include <XMLHelper.h>`

Public Member Functions

- **Point** ()
  Default constructor.
- **Point** (const **Point** &p)
  Copy constructor.
- void **set** (double x, double y)
  Mutator.
- double **x** () const
  Accessor for the x-coordinate.
- double **y** () const
  Accessor for the y-coordinate.
- **Point** & **operator**= (const **Point** &p)
  Assignment operator.
- bool **operator**< (const **Point** &p) const
  Less-than operator.
- bool **operator**== (const **Point** &p) const
  Equality operator.

Private Attributes

- double **mX**
  The x coordinate.
- double **mY**
  The y coordinate.

4.106.1 Detailed Description

Class representing a point. Used when parsing Polygons.

Author:
Per Alexius

Date **Date**
2007/01/24 13:13:27

4.106.2 Constructor & Destructor Documentation

4.106.2.1 Point::Point (const Point & *p*) [inline]

Copy constructor.

Parameters:
p The Point to copy.

4.106.3 Member Function Documentation

4.106.3.1 bool Point::operator< (const Point & *p*) const [inline]

Less-than operator.

Parameters:
p The Point to compare with.

Returns:
True if this Point is less than the provided Point.

4.106.3.2 Point& Point::operator= (const Point & *p*) [inline]

Assignment operator.

Parameters:
p The Point to assign to this point.

Returns:
The assigned Point.

4.106.3.3 bool Point::operator== (const Point & *p*) const [inline]

Equality operator.

Parameters:
p The Point to compare with.

Returns:
True if this Point is equal to the provided Point.

4.106.3.4 void Point::set (double *x*, double *y*) [inline]

Mutator.

Parameters:
x The x-coordinate.
y The y-coordinate.

4.106.3.5 double Point::x () const [inline]

Accessor for the x-coordinate.

Returns:
x The x-coordinate.

4.106.3.6 double Point::y () const [inline]

Accessor for the y-coordinate.

Returns:
y The y-coordinate.

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

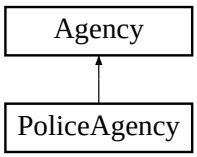
- XMLHelper.h

4.107 PoliceAgency Class Reference

Class containing functionality for controlling PoliceAgencyTeams.

`#include <Agency.h>`

Inheritance diagram for PoliceAgency::



Public Member Functions

- **PoliceAgency** (const std::vector< **AgencyTeam** * > &teams, **Grid** &g)
  Constructor.
- void **setTeamsGoals** ()
  Determine positions of the teams.

Private Member Functions

- bool **severeProblem** ()
  Determines if we have a severe problem. If we do - then the weights for the clustering algorithm is set.

4.107.1 Detailed Description

Class containing functionality for controlling PoliceAgencyTeams.

Author:
Per Alexius

Date:
2006/10/02 16:01:25

4.107.2 Constructor & Destructor Documentation

4.107.2.1 PoliceAgency::PoliceAgency (const std::vector< AgencyTeam * > & *teams*, Grid & *g*) `[inline]`

Constructor.

Parameters:
teams A vector containing this Agency's teams.
g A reference to the **Grid**(p. 227).

4.107.3 Member Function Documentation

4.107.3.1 void PoliceAgency::setTeamsGoals () [virtual]

Determine positions of the teams.

Assigns one team to each cluster found. If there are more teams than clusters - assign more than one team to each cluster.

Reimplemented from **Agency** (p. 28).

4.107.3.2 bool PoliceAgency::severeProblem () [private, virtual]

Determines if we have a severe problem. If we do - then the weights for the clustering algorithm is set.

Returns:
True if we have a severe violence problem, false otherwise.

Implements **Agency** (p. 28).

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

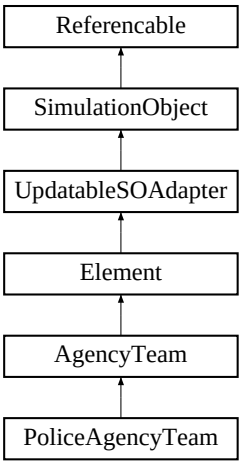
- Agency.h
- Agency.cpp

4.108 PoliceAgencyTeam Class Reference

Class representing a PoliceAgencyTeam.

`#include <AgencyTeam.h>`

Inheritance diagram for PoliceAgencyTeam::



Public Member Functions

- **PoliceAgencyTeam** (const **DataObject** &d)

  Constructor that creates an **AgencyTeam**(p. 32) *from the provided* **DataObject**(p. 145).

- void **act** (**Time** now)

  Performs this teams actions.

4.108.1 Detailed Description

Class representing a PoliceAgencyTeam.

Author:
Per Alexius

Date **Date**
2006/10/10 09:35:59

4.108.2 Constructor & Destructor Documentation

4.108.2.1 PoliceAgencyTeam::PoliceAgencyTeam (const DataObject & *d*) [inline]

Constructor that creates an **AgencyTeam**(p. 32) from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.108.3 Member Function Documentation

4.108.3.1 void PoliceAgencyTeam::act (Time *now*) [virtual]

Performs this teams actions.

Reduce violence in all cells in this team's area of influence until the team's capacity limit is met.

Parameters:

now The current simulation time.

Implements **AgencyTeam** (p. 35).

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

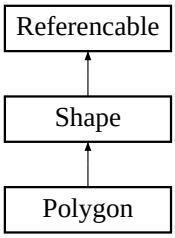
- AgencyTeam.h
- AgencyTeam.cpp

4.109 Polygon Class Reference

A class representing a Polygon.

`#include <Shape.h>`

Inheritance diagram for Polygon::



Public Member Functions

- **Polygon** ()
- **Polygon** (const **Reference** &ref)
- **Polygon** (const gpc_polygon &p, std::list< std::string > lineId, const **Reference** &ref)

  Constructor that creates a Polygon based on a gpc_ polygon.

- **Polygon** (**LatLng** cen, double ang, double w, double h, const **Reference** &r)

  Creates a rectangle with the specified center, angle, width and height.

- **Polygon** (const **Polygon** &p)

  Copy constructor.

- void **toProj** (const **Projection** &proj)

  Projects this **Shape**(p. 412) *using the specified* **Projection**(p. 348).

- void **toCoord** (const **Projection** &proj)

  Transforms this **Shape**(p. 412) *to lat lng coordinate using the provided projection.*

- void **cells** (const **BasicGrid** &g, std::list< **GridPos** > &outCells) const

  Returns a list containing pointers to all cells covered by this **Shape**(p. 412).

- **LatLng cenCoord** () const

  Returns the center coordinate in lat lng of this **Shape**(p. 412).

- **ProjCoord cenProj** () const

  Returns the center coordinate in projection space of this **Shape**(p. 412).

- void **boundingBox** (double &t, double &l, double &b, double &r) const

  Gets the bounding box of this **Shape**(p. 412).

- double **area** () const

  Returns the area of this **Shape**(p. 412). *Not yet implemented for Polygons.*

- void **move** (double dx, double dy)

  Moves this **Shape**(p. 412) *relative to itself.*

- void **move** (**LatLng** newPos)

  Moves this **Shape**(p. 412) *to a new position.*

- **Shape** * **clone** () const

  Creates a deep copy of this **Shape**(p. 412).

- const std::string **type** () const

  Returns the stratmas protocol type of this shape.

- const gpc_polygon & **boundary** () const

  Accessor for the gpc_polygon that constitutes this Polygon.

- std::ostream & **toXML** (std::ostream &o, std::string indent) const

  Writes an XML representation of this object to the provided stream with nice indentation.

Static Public Member Functions

- void **deallocGpcPolygon** (gpc_polygon &p)

  Deallocates the memory used by a gpc_polygon.

- void **deepCopyGpcPolygon** (gpc_polygon &dst, const gpc_polygon &src)

  Performs a deep copy of a gpc_polygon.

Protected Attributes

- **LatLng mCenter**

  Center of the Polygon's bounding box.

- gpc_polygon **mBoundary**

  The Polygon's boundary is represented as a gpc_polygon.

- std::list< std::string > **mLineId**

  List containing the identifiers of the lines.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Polygon** &p)

  For debugging purposes.

4.109.1 Detailed Description

A class representing a Polygon.

Author:
Per Alexius

Date:
2006/07/19 07:04:39

4.109.2 Constructor & Destructor Documentation

4.109.2.1 Polygon::Polygon ()

Default constructor.

4.109.2.2 Polygon::Polygon (const Reference & *ref*)

Creates a Polygon with the specified **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to this Polygon.

4.109.2.3 Polygon::Polygon (const gpc_polygon & *p*, std::list< std::string > *lineId*, const Reference & *ref*)

Constructor that creates a Polygon based on a gpc_polygon.

Assumes that p is given in lat lng.

Parameters:
p The gpc_polygon to build this Polygon from.

lineId A vector containing the id:s of the lines that constitutes this Polygon. Ordered in the same order as the points in the provided gpc_polygon.

ref The **Reference**(p. 378) to this Polygon.

4.109.2.4 Polygon::Polygon (LatLng *cen*, double *ang*, double *w*, double *h*, const Reference & *r*)

Creates a rectangle with the specified center, angle, width and height.

Parameters:
cen The center of the rectangle.

ang The angle in which the rectangle leans.

w The width of the rectangle in meters.

h the height of the rectangle in meters.

r The reference to the shape to be created.

4.109.2.5 Polygon::Polygon (const Polygon & *p*)

Copy constructor.

Parameters:
p The Polygon to copy.

4.109.3 Member Function Documentation

4.109.3.1 double Polygon::area () const [inline, virtual]

Returns the area of this **Shape**(p. 412). Not yet implemented for Polygons.

Returns:
The area of this **Shape**(p. 412).

Implements **Shape** (p. 414).

4.109.3.2 const gpc_polygon& Polygon::boundary () const [inline]

Accessor for the gpc_polygon that constitutes this Polygon.

Returns:
The gpc_polygon that constitutes this Polygon.

4.109.3.3 void Polygon::boundingBox (double & *t*, double & *l*, double & *b*, double & *r*) const [virtual]

Gets the bounding box of this **Shape**(p. 412).

Parameters:
t Top coordinate of this Shape's boundingbox.
l Left coordinate of this Shape's boundingbox.
b Bottom coordinate of this Shape's boundingbox.
r Right coordinate of this Shape's boundingbox.

Implements **Shape** (p. 414).

4.109.3.4 void Polygon::cells (const BasicGrid & *g*, std::list< GridPos > & *outCells*) const [virtual]

Returns a list containing pointers to all cells covered by this **Shape**(p. 412).

Parameters:
g A reference to the **Grid**(p. 227).
outCells A list that on return contains pointers to all cells covered by this **Shape**(p. 412).

Implements **Shape** (p. 414).

4.109.3.5 LatLng Polygon::cenCoord () const [inline, virtual]

Returns the center coordinate in lat lng of this **Shape**(p. 412).

Returns:
The center coordinate of this **Shape**(p. 412).

Implements **Shape** (p. 415).

4.109.3.6 ProjCoord Polygon::cenProj () const [inline, virtual]

Returns the center coordinate in projection space of this **Shape**(p. 412).

Returns:
The center coordinate of this **Shape**(p. 412).

Implements **Shape** (p. 415).

4.109.3.7 Shape* Polygon::clone () const [inline, virtual]

Creates a deep copy of this **Shape**(p. 412).

Returns:
A newly allocated copy of this **Shape**(p. 412).

Implements **Shape** (p. 416).

4.109.3.8 void Polygon::deallocGpcPolygon (gpc_polygon & *p*) [static]

Deallocates the memory used by a gpc_polygon.

Notice that the gpc_polygon struct won't be deallocated itself, but only it's contents. This leaves the possibility to deallocate partly statically allocated gpc_polygons.

Parameters:
p The gpc_polygon to be deallocated

4.109.3.9 void Polygon::deepCopyGpcPolygon (gpc_polygon & *dst*, const gpc_polygon & *src*) [static]

Performs a deep copy of a gpc_polygon.

Parameters:
dst The destination gpc_polygon

src The source gpc_polygon

4.109.3.10 void Polygon::move (LatLng *newPos*) [virtual]

Moves this **Shape**(p. 412) to a new position.

Parameters:
newPos The position to move to.

Implements **Shape** (p. 416).

4.109.3.11 void Polygon::move (double *dx*, double *dy*) [virtual]

Moves this **Shape**(p. 412) relative to itself.

Parameters:
dx The movement in x-direction in degrees longitude
dy The movement in y-direction in degrees latitude

Implements **Shape** (p. 416).

4.109.3.12 void Polygon::toCoord (const Projection & *proj*) [virtual]

Transforms this **Shape**(p. 412) to lat lng coordinate using the provided projection.

Parameters:
proj The projection to use.

Implements **Shape** (p. 416).

4.109.3.13 void Polygon::toProj (const Projection & *proj*) [virtual]

Projects this **Shape**(p. 412) using the specified **Projection**(p. 348).

Parameters:
proj The projection to use.

Implements **Shape** (p. 416).

4.109.3.14 ostream & Polygon::toXML (std::ostream & *o*, std::string *indent*) const [virtual]

Writes an XML representation of this object to the provided stream with nice indentation.

Parameters:
o The stream to write to.
indent Indentation string.

Returns:
The stream with the xml representation written to it.

Implements **Shape** (p. 417).

4.109.3.15 const std::string Polygon::type () const [inline, virtual]

Returns the stratmas protocol type of this shape.

Returns:
The stratmas protocol type of this shape.

Implements **Shape** (p. 417).

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

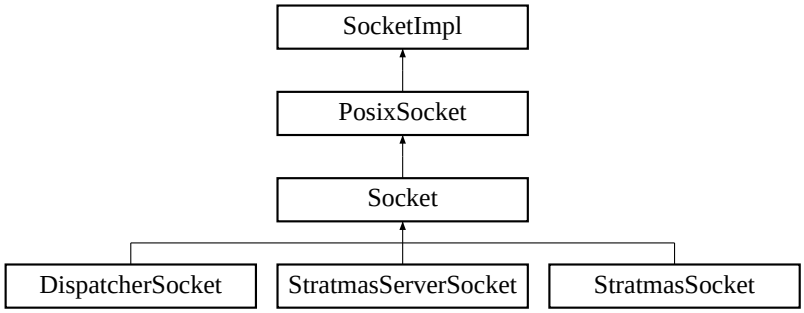
- Shape.h
- Shape.cpp

4.110 PosixSocket Class Reference

C++ wrapper around a posix socket.

`#include <PosixSocket.h>`

Inheritance diagram for PosixSocket::



Public Member Functions

- **PosixSocket** ()
  Constructor.
- virtual ~**PosixSocket** ()
  Destructor.
- virtual bool **create** ()
  Creates the underlying socket.
- virtual bool **bind** (const char *host, int port)
  Binds this socket to the provided address and port.
- virtual bool **listen** () const
  Listens to connections.
- virtual bool **accept** (**Socket** &newSock) const
  Accepts a connection.
- virtual bool **connect** (const std::string host, const int port)
  Connects to the specified port on the specified host.
- virtual bool **close** ()
  Closes the socket.
- virtual bool **send** (const void *msg, unsigned int len) const
  Sends data over the socket.
- virtual int **recv** (void *msg, unsigned int len) const
  Receives data from the socket.

- virtual int **recvf** (void *msg, int len) const

  Receives an exact amount of data from the socket.

- virtual void **set_non_blocking** (const bool b)

  Set the O_NONBLOCK flag.

- virtual bool **valid** () const

  Checks if this socket is valid.

- virtual std::string **address** () const

  Returns a string representation of the address of this socket.

Protected Attributes

- int **mSock**

  The file descriptor for the socket.

- sockaddr_in **mAddr**

  The name assigned to this socket.

4.110.1 Detailed Description

C++ wrapper around a posix socket.

Author:
Per Alexius

Date:
2006/07/03 14:18:23

4.110.2 Member Function Documentation

4.110.2.1 bool PosixSocket::accept (Socket & *newSock*) const [virtual]

Accepts a connection.

Parameters:
newSock On return - the socket from which the connection was accepted.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.110.2.2 std::string PosixSocket::address () const [virtual]

Returns a string representation of the address of this socket.

Returns:
A string representation of the address of this socket.

Implements **SocketImpl** (p. 435).

4.110.2.3 bool PosixSocket::bind (const char * *host*, int *port*) [virtual]

Binds this socket to the provided address and port.

Parameters:
host The name of the host or null if INADDR_ANY should be used.
port The port.

Returns:
True if the socket was successfully bound, false otherwise.

Implements **SocketImpl** (p. 435).

4.110.2.4 bool PosixSocket::close () [virtual]

Closes the socket.

Returns:
True on success.

Implements **SocketImpl** (p. 435).

4.110.2.5 bool PosixSocket::connect (const std::string *host*, const int *port*) [virtual]

Connects to the specified port on the specified host.

Parameters:
host The host to connect to.
port The port to connect to.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.110.2.6 bool PosixSocket::create () [virtual]

Creates the underlying socket.

Returns:
True if the socket was successfully created, false otherwise.

Implements **SocketImpl** (p. 435).

4.110.2.7 bool PosixSocket::listen () const [virtual]

Listens to connections.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.110.2.8 int PosixSocket::recv (void * *msg*, unsigned int *len*) const [virtual]

Receives data from the socket.

Parameters:
msg On return - the data received.

len The maximum length in bytes of the data to receive.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.110.2.9 int PosixSocket::recvf (void * *msg*, int *len*) const [virtual]

Receives an exact amount of data from the socket.

Parameters:
msg On return - the data received.

len The number of bytes to receive.

Returns:
The total number of bytes read.

Implements **SocketImpl** (p. 435).

4.110.2.10 bool PosixSocket::send (const void * *msg*, unsigned int *len*) const [virtual]

Sends data over the socket.

Parameters:
msg The data to send.

len The length in bytes of the data to be sent.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.110.2.11 void PosixSocket::set_non_blocking (const bool *b*) [virtual]

Set the O_NONBLOCK flag.

Parameters:
b New value of the O_NONBLOCK flag.

Implements **SocketImpl** (p. 435).

4.110.2.12 bool PosixSocket::valid () const [virtual]

Checks if this socket is valid.

Returns:
True if this socket is valid, false otherwise.

Implements **SocketImpl** (p. 435).

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

- PosixSocket.h
- PosixSocket.cpp

4.111 PresenceObject Class Reference

PresenceObjects are used to mark units' presence in grid cells. This information is then used by the combat model.

`#include <PresenceObject.h>`

Public Member Functions

- **PresenceObject** (int c, **Unit** &u, double f)

  Create a PresenceObject with the specified properties.

- void **set** (int c, **Unit** &u, double f)

  Sets the properties of a PresenceObject.

- int **cell** () const

  Accessor for the cell index.

- **Unit** & **unit** () const

  Accessor for the **Unit**(p. 546).

- double **fraction** () const

  Accessor for the fraction.

- void **affect** (std::vector< **PresenceObject** * > &potentialVictims) const

  Lets the **Unit**(p. 546) *this PresenceObject refers to affect the units in the PresenceObjects in the provided vector.*

- bool **operator<** (const **PresenceObject** &p) const

  Less than operator.

Private Attributes

- int **mCell**

  The cell this PresenceObject refers to.

- **Unit** * **mUnit**

  The **Unit**(p. 546) *this PresenceObject refers to.*

- double **mFraction**

  The fraction of the unit present in the cell this PresenceObject refers to.

4.111.1 Detailed Description

PresenceObjects are used to mark units' presence in grid cells. This information is then used by the combat model.

Author:
Per Alexius

Date:
2006/03/06 12:55:11

4.111.2 Constructor & Destructor Documentation

4.111.2.1 PresenceObject::PresenceObject (int *c*, Unit & *u*, double *f*) [inline]

Create a PresenceObject with the specified properties.

Parameters:
c The cell index.

u The **Unit**(p. 546).

f The fraction of the **Unit**(p. 546).

4.111.3 Member Function Documentation

4.111.3.1 void PresenceObject::affect (std::vector< PresenceObject * > & *potentialVictims*) const

Lets the **Unit**(p. 546) this PresenceObject refers to affect the units in the PresenceObjects in the provided vector.

Called by the **CombatGrid**(p. 101) after finding out which units that overlaps which cells.

Parameters:
potentialVictims Vector with potential victims.

4.111.3.2 int PresenceObject::cell () const [inline]

Accessor for the cell index.

Returns:
The cell's index.

4.111.3.3 double PresenceObject::fraction () const [inline]

Accessor for the fraction.

Returns:
The fraction.

4.111.3.4 bool PresenceObject::operator< (const PresenceObject & *p*) const [inline]

Less than operator.

A PresenceObject p1 is less than another PresenceObject p2 if:

p1's cell index is less than p2's

or

p1 and p2 have the same cell index and p1's unit pointer is less than p2's.

Parameters:

p The PresenceObject to compare to.

Returns:

True if this PresenceObject is less than p.

4.111.3.5 void PresenceObject::set (int *c*, Unit & *u*, double *f*) [inline]

Sets the properties of a PresenceObject.

Parameters:

c The cell index.

u The **Unit**(p. 546).

f The fraction of the **Unit**(p. 546).

4.111.3.6 Unit& PresenceObject::unit () const [inline]

Accessor for the **Unit**(p. 546).

Returns:

The **Unit**(p. 546).

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

- PresenceObject.h
- PresenceObject.cpp

4.112 PresenceObjectAllocator Class Reference

Helper class that makes the memory allocation for PresenceObjects more effective.

`#include <PresenceObjectAllocator.h>`

Public Member Functions

- **~PresenceObjectAllocator** ()

  Destructor.

- **PresenceObject** * **create** (int c, **Unit** &u, double f)

  Provides a **PresenceObject**(p. 338) *with the specified properties either by using an already allocated* **PresenceObject**(p. 338) *or by allocating a new one.*

- void **dismiss** (**PresenceObject** *p)

  Notifies the PresenceObjectAllocator that the provided **PresenceObject**(p. 338) *will no longer be used and that it thus may be considered freed.*

- void **reset** ()

  Resets the PresenceObjectAllocator by deallocating all memory used.

Private Attributes

- std::set< **PresenceObject** * > **mUsed**

  Allocated and used PresenceObjects.

- std::set< **PresenceObject** * > **mFree**

  Allocated and unused PresenceObjects.

4.112.1 Detailed Description

Helper class that makes the memory allocation for PresenceObjects more effective.

Author:
Per Alexius

Date:
2006/03/02 17:06:54

4.112.2 Member Function Documentation

4.112.2.1 PresenceObject * PresenceObjectAllocator::create (int *c*, Unit & *u*, double *f*)

Provides a **PresenceObject**(p. 338) with the specified properties either by using an already allocated **PresenceObject**(p. 338) or by allocating a new one.

Parameters:

c The cell index.

u The **Unit**(p. 546).

f The fraction of the **Unit**(p. 546).

Returns:

A **PresenceObject**(p. 338) with the specified properties.

4.112.2.2 void PresenceObjectAllocator::dismiss (PresenceObject * *p*)

Notifies the PresenceObjectAllocator that the provided **PresenceObject**(p. 338) will no longer be used and that it thus may be considered freed.

Parameters:

p The PresenceObjec to dismiss.

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

- PresenceObjectAllocator.h
- PresenceObjectAllocator.cpp

4.113 PrivateRandom Class Reference

Helper class for handling random numbers that should not interfere with the sequence of random numbers generated during a simlation.

`#include <random.h>`

Static Public Member Functions

- void **initRandomNumberArray** ()

  Initializes an array of random numbers for later use.

- long **privateRandomUniform** ()

  Returns a random long that is uniformly distributed between zero and kRandomMax. This function does not influence the order of random numbers generated during a simulation.

Static Private Attributes

- const int **kNumRand** = 10000

  Size of array of random numbers that may not affect random number repeatability.

- int **sRandIndex**

  Current index in the random number array.

- long **sRandNum** [**kNumRand**]

  Array of random numbers allowing generation of random numbers that does not affect random number repeatability.

4.113.1 Detailed Description

Helper class for handling random numbers that should not interfere with the sequence of random numbers generated during a simlation.

Author:
Per Alexius

Date **Date**
2006/09/05 14:18:21

4.113.2 Member Function Documentation

4.113.2.1 void PrivateRandom::initRandomNumberArray () [inline, static]

Initializes an array of random numbers for later use.

Should be called once (and only once) at startup.

4.113.2.2 long PrivateRandom::privateRandomUniform () [inline, static]

Returns a random long that is uniformly distributed between zero and kRandomMax. This function does not influence the order of random numbers generated during a simulation.

This function exists due to the fact that Shapes use random numbers to keep track of whether they have changed or not and the sequence of **Shape**(p. 412) creation may differ from one simulation to another (depending on subscriptions for example). If Shapes gets random numbers from **RandomUniform()**(p. 611) the sequence may not be repeatable.

Returns:
A random undigned long that is uniformly distributed between zero and kRandomMax.

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

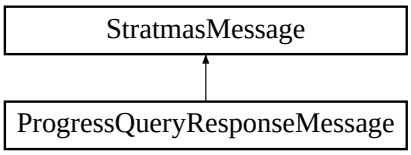
- **random.h**
- **stratmas.cpp**

4.114 ProgressQueryResponseMessage Class Reference

Class representing the ProgressQueryResponseMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for ProgressQueryResponseMessage::



Public Member Functions

- **ProgressQueryResponseMessage** (**Buffer** &b)
  Creates a ProgressQueryResponseMessage that fetches its data from the specified **Buffer**(p. 67).
- virtual ~**ProgressQueryResponseMessage** ()
  Destructor.
- void **toXML** (std::ostream &o) const
  Produces the XML representation of this message.

Private Attributes

- **Buffer** & **mBuf**
  The **Buffer**(p. 67) *from which data should be fetched.*

4.114.1 Detailed Description

Class representing the ProgressQueryResponseMessage.

Author:
Per Alexius

Date **Date**
2006/03/06 14:23:12

4.114.2 Constructor & Destructor Documentation

4.114.2.1 ProgressQueryResponseMessage::ProgressQueryResponseMessage (Buffer & *b*) [inline]

Creates a ProgressQueryResponseMessage that fetches its data from the specified **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to fetch data from.

4.114.3 Member Function Documentation

4.114.3.1 void ProgressQueryResponseMessage::toXML (std::ostream & *o*) const [virtual]

Produces the XML representation of this message.

Parameters:

o The stream to which the message is written

Implements **StratmasMessage** (p. 473).

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

- StratmasMessage.h
- StratmasMessage.cpp

4.115 ProjCoord Class Reference

A class representing a coordinate in projection space.

`#include <ProjCoord.h>`

Public Member Functions

- **ProjCoord** (double x, double y)
  Obvious constructor.
- void **set** (double inx, double iny)
  Sets the coordinate to (inx, iny).
- double **x** () const
  Returns the x value of this coordinate.
- double **y** () const
  Returns the y value of this coordinate.
- double **squDistanceTo** (const **ProjCoord** &p) const
  Returns the square of the distance (in meters) between this point and p.
- **LatLng toLatLng** () const
  Convert this projected point back to lat, lng.

Protected Attributes

- double **mX**
  X-value of coordinate.
- double **mY**
  Y-value of coordinate.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **ProjCoord** &p)
  For debugging purposes.

4.115.1 Detailed Description

A class representing a coordinate in projection space.

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

- ProjCoord.h
- ProjCoord.cpp

4.116 Projection Class Reference

This class holds data related to the projection used when partitioning the **Grid**(p. 227).

`#include <Projection.h>`

Public Member Functions

- **Projection** (double R, double cenLat, double cenLng)

  Constructor.

- virtual ~**Projection** ()

  Destructor.

- void **setProjCenter** (double inX, double inY)

  Sets the center of the projection.

- void **coordToProj** (const double inLng, const double inLat, double &outX, double &outY) const

  Projects a lat, lng coordinate on the projection surface.

- void **projToCoord** (const double inX, const double inY, double &outLng, double &outLat) const

  Converets a projected point back to lat, lng.

- **ProjCoord coordToProj** (const **LatLng** &l) const

  Projects a **LatLng**(p. 271) *coordinate on the projection surface.*

- **LatLng projToCoord** (const **ProjCoord** &p) const

  Converts a projected coordinate back to lat, lng.

- void **coordToProj** (gpc_vertex &dst, const gpc_vertex &src) const

  Helper for gpc-related projecting.

- void **coordToProj** (gpc_vertex_list &dst, const gpc_vertex_list &src) const

  Helper for gpc-related projecting.

- void **coordToProj** (gpc_polygon &dst, const gpc_polygon &src) const

  Helper for gpc-related projecting.

- void **projToCoord** (gpc_vertex &dst, const gpc_vertex &src) const

  Helper for gpc-related projecting.

- void **projToCoord** (gpc_vertex_list &dst, const gpc_vertex_list &src) const

  Helper for gpc-related projecting.

- void **projToCoord** (gpc_polygon &dst, const gpc_polygon &src) const

  Helper for gpc-related projecting.

Static Public Member Functions

- const **Projection** * **currentProjection** ()

  Returns the current projection.

Static Public Attributes

- **Projection** * **mCurrent** = 0

  Holds a pointer to the current projection used by the simulation.

Private Member Functions

- **Projection** ()

  Private default constructor.

- void **setCoordCenter** (double inLat, double inLng)

  Sets the center coordinate and the projection parameters related to it.

Private Attributes

- double **mPCenX**

  x-coordinate of the center of the projection

- double **mPCenY**

  y-coordinate of the center of the projection

- double **mCenLng**

  Longitude of center of area to be projected.

- double **mCenLat**

  Latitude of center of area to be projected.

- double **mCosPhi0**

  Projection parameter.

- double **mSinPhi0**

  Projection parameter.

- double **mR**

  Radius of sphere to be projected upon.

- double **mPhi0**

  Projection parameter.

- double **mLam0**

  Projection parameter.

4.116.1 Detailed Description

This class holds data related to the projection used when partitioning the **Grid**(p. 227).

Author:
Per Alexius

Date:
2005/06/13 11:19:06

4.116.2 Constructor & Destructor Documentation

4.116.2.1 Projection::Projection (double *R*, double *cenLat*, double *cenLng*)

Constructor.

Parameters:
R Radius of sphere to be used as projection surface. The simulation sets it to the earth radius in order to get projection units in meters.
cenLat Latitude of center of area to be projected.
cenLng Longitude of center of area to be projected.

4.116.3 Member Function Documentation

4.116.3.1 void Projection::coordToProj (gpc_vertex_list & *dst*, const gpc_vertex_list & *src*) const

Helper for gpc-related projecting.

Parameters:
dst The destination gpc_vertex_list
src The source gpc_vertex_list

4.116.3.2 void Projection::coordToProj (gpc_vertex & *dst*, const gpc_vertex & *src*) const

Helper for gpc-related projecting.

Parameters:
dst The destination gpc_vertex
src The source gpc_vertex

4.116.3.3 ProjCoord Projection::coordToProj (const LatLng & *l*) const

Projects a **LatLng**(p. 271) coordinate on the projection surface.

Parameters:
l Coordinate to be projected

Returns:
The projected coordinate.

4.116.3.4 void Projection::coordToProj (const double *inLng*, const double *inLat*, double & *outX*, double & *outY*) const

Projects a lat, lng coordinate on the projection surface.

Parameters:

inLng Longitude of in coordinate

inLat Latitude of in coordinate

outX x-coordinate of projected point

outY y-coordinate of projected point

4.116.3.5 const Projection* Projection::currentProjection () [inline, static]

Returns the current projection.

Returns:

The current projection.

4.116.3.6 LatLng Projection::projToCoord (const ProjCoord & *p*) const

Converts a projected coordinate back to lat, lng.

Parameters:

p Coordinate to be converted

Returns:

LatLng(p. 271) coordinate.

4.116.3.7 void Projection::projToCoord (const double *inX*, const double *inY*, double & *outLng*, double & *outLat*) const

Converets a projected point back to lat, lng.

Parameters:

inX x-coordinate of point to be converted

inY y-coordinate of point to be converted

outLng Resulting longitude

outLat Resulting latitude

4.116.3.8 void Projection::setCoordCenter (double *inLat*, double *inLng*) [private]

Sets the center coordinate and the projection parameters related to it.

Parameters:

inLat Latitude of center of area to be projected.

inLng Longitude of center of area to be projected.

4.116.3.9 void Projection::setProjCenter (double *inX*, double *inY*) [inline]

Sets the center of the projection.

Parameters:

inX The center x coordinate.

inY The center y coordinate.

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

- Projection.h
- Projection.cpp

4.117 PropertyHandler Class Reference

Handles different properties that may be set for the server.

`#include <PropertyHandler.h>`

Static Public Member Functions

- bool **unitRandomWalk** ()

  Accessor for the unitRandomWalk property.

- bool **validateXML** ()

  Accessor for the validateXML property.

- bool **setPropertiesFromFile** (const std::string &filename)

  Gets properties on the form 'propertyname' = 'value' from the specified file.

- void **setProperty** (std::string property, std::string value)

  Sets the specified property to the specifued value.

Static Private Member Functions

- bool **stringToBool** (std::string &value)

  Converts a string to a bool value.

Static Private Attributes

- bool **mUnitRandomWalk** = false

  Set to true if units should move randomly [deafult false].

- bool **mValidateXML** = true

  Set to false if xml should not be validated [deafult true].

4.117.1 Detailed Description

Handles different properties that may be set for the server.

Mostly used for debuging purposes.

Author:
Per Alexius

Date:
2006/03/06 14:23:09

4.117.2 Member Function Documentation

4.117.2.1 bool PropertyHandler::setPropertiesFromFile (const std::string & *filename*) [static]

Gets properties on the form 'propertyname' = 'value' from the specified file.

Parameters:

filename The name of the file.

Returns:

True if the file was read successfully, false otherwise.

4.117.2.2 void PropertyHandler::setProperty (std::string *property*, std::string *value*) [static]

Sets the specified property to the specifued value.

Parameters:

property The name of the property to set.

value The value to use.

4.117.2.3 bool PropertyHandler::stringToBool (std::string & *value*) [static, private]

Converts a string to a bool value.

Parameters:

value The string to convert.

Returns:

The bool value of the provided string.

4.117.2.4 bool PropertyHandler::unitRandomWalk () [inline, static]

Accessor for the unitRandomWalk property.

Returns:

The value of the unitRandomWalk property.

4.117.2.5 bool PropertyHandler::validateXML () [inline, static]

Accessor for the validateXML property.

Returns:

The value of the validateXML property.

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

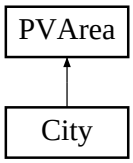
- PropertyHandler.h
- PropertyHandler.cpp

4.118 PVArea Class Reference

This class represents the interface for modification of pv variables in an area.

`#include <PVArea.h>`

Inheritance diagram for PVArea::



Public Member Functions

- virtual ~**PVArea** ()
  Destructor.
- virtual const **Shape** & **area** () const =0
  Accessor for the **Shape**(p. 412) *the modifications refer to.*
- virtual const **Distribution** & **distribution** () const =0
  Accessor for the **Distribution**(p. 173) *of the modifications over the area.*
- const std::vector< **PVModification** > & **pvs** () const
  Accessor for the modifications.

Protected Attributes

- std::vector< **PVModification** > **mPVs**
  The modifications.

4.118.1 Detailed Description

This class represents the interface for modification of pv variables in an area.

Author:
Per Alexius

Date **Date**
2006/07/19 07:04:34

4.118.2 Member Function Documentation

4.118.2.1 virtual const Shape& PVArea::area () const [pure virtual]

Accessor for the **Shape**(p. 412) the modifications refer to.

Returns:
The **Shape**(p. 412).

Implemented in **City** (p. 92).

4.118.2.2 virtual const Distribution& PVArea::distribution () const [pure virtual]

Accessor for the **Distribution**(p. 173) of the modifications over the area.

Returns:
The **Distribution**(p. 173).

Implemented in **City** (p. 92).

4.118.2.3 const std::vector<PVModification>& PVArea::pvs () const [inline]

Accessor for the modifications.

Returns:
The modifications.

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

- PVArea.h

4.119 PVDescription Class Reference

This class contains the description of a process variable.

`#include <PVInfo.h>`

Public Member Functions

- **PVDescription** (std::string n, std::string type, std::string c, bool f, std::string min, std::string max, bool v=true)

  Creates a PVDescription from the provided values.

- **PVDescription** (const **PVDescription** &p)

  Copy constructor.

- bool **visible** () const

  Accessor for the visible flag.

- bool **operator<** (const **PVDescription** &p)

  Less-than operator.

- std::ostream & **toXML** (std::ostream &o) const

  Produces an XML representation of this PVDescription according to the xml schemas.

Private Attributes

- std::string **mName**

  The name of the PV.

- std::string **mType**

  The type of the PV.

- std::string **mCategory**

  The category of the PV.

- bool **mFactions**

  Flag for factions or not.

- std::string **mMin**

  The minimum value.

- std::string **mMax**

  The maximum value.

- bool **mVisible**

  Flag indicating user visibility.

4.119.1 Detailed Description

This class contains the description of a process variable.

Author:
Per Alexius

Date:
2007/01/24 13:13:23

4.119.2 Constructor & Destructor Documentation

4.119.2.1 PVDescription::PVDescription (std::string *n*, std::string *type*, std::string *c*, bool *f*, std::string *min*, std::string *max*, bool *v* = true) [inline]

Creates a PVDescription from the provided values.

Parameters:
n The name of the PV.

c The category of the PV.

f Flag for factions or not.

min The minimum value.

max The maximum value.

v Flag indicating user visibility.

4.119.2.2 PVDescription::PVDescription (const PVDescription & *p*) [inline]

Copy constructor.

Parameters:
p The PVDescription to copy.

4.119.3 Member Function Documentation

4.119.3.1 bool PVDescription::operator< (const PVDescription & *p*) [inline]

Less-than operator.

Parameters:
p The PVDescription to compare with.

Returns:
True if this PVDescription is less than the provided PVDescription.

4.119.3.2 ostream & PVDescription::toXML (std::ostream & *o*) const

Produces an XML representation of this PVDescription according to the xml schemas.

Parameters:
o The ostream to print to.

Returns:
The ostream with the XML representation written to it.

4.119.3.3 bool PVDescription::visible () const [inline]

Accessor for the visible flag.

Returns:
The value of the visible flag.

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

- PVInfo.h
- PVInfo.cpp

4.120 PVHelper Class Reference

Helper class for handling mapping between pv indices, names and types etc.

`#include <ProcessVariables.h>`

Static Public Member Functions

- const char * **pvfName** (ePVF pv)
- const char * **pvfType** (ePVF pv)
- const char * **pvName** (ePV pv)
- const char * **pvType** (ePV pv)
- const char * **pdfName** (eDerivedF pv)
- const char * **pdfType** (eDerivedF pv)
- const char * **pdName** (eDerived pv)
- const char * **pdType** (eDerived pv)
- const char * **pcfName** (ePreCalcF pv)
- const char * **pcfType** (ePreCalcF pv)
- const char * **pcName** (ePreCalc pv)
- const char * **pcType** (ePreCalc pv)
- const char * **modifiablePVName** (int pv)
- const char * **allPVName** (eAllPV pv)
- eAllPV **nameToOverAllOrder** (const std::string &name)

  Maps an attribute name to its order in the attribute array.

- eAllPV **displayNameToOverAllOrder** (const std::string &name)

  Maps an attribute's display name to its order in the attribute array.

Static Private Attributes

- std::map< std::string, eAllPV > **sNameToOverAllOrder**

  Maps a PV name to its order in the eAllPV enumeration.

- std::map< std::string, eAllPV > **sDisplayNameToOverAllOrder**

  Maps a PV display name to its order in the eAllPV enumeration.

4.120.1 Detailed Description

Helper class for handling mapping between pv indices, names and types etc.

Author:
Per Alexius

Date **Date**
2007/01/24 13:13:25

4.120.2 Member Function Documentation

4.120.2.1 eAllPV PVHelper::displayNameToOverAllOrder (const std::string & *name*) [static]

Maps an attribute's display name to its order in the attribute array.

Parameters:
name The attribute name.

Returns:
The value for the specified attribute.

4.120.2.2 eAllPV PVHelper::nameToOverAllOrder (const std::string & *name*) [static]

Maps an attribute name to its order in the attribute array.

Parameters:
name The attribute name.

Returns:
The value for the specified attribute.

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

- ProcessVariables.h
- ProcessVariables.cpp

4.121 PVInfo Class Reference

Static class that holds information about the process variables that the server is capable of simulating.

`#include <PVInfo.h>`

Static Public Member Functions

- void **init** ()

  Initializes this PVInfo object.

- void **addPV** (std::string n, std::string t, std::string c, bool f, std::string min, std::string max)

  Adds a **PVDescription**(p. 357) *with the provided values.*

- void **reset** ()

  Resets this PVInfo object.

- std::ostream & **toXML** (std::ostream &o)

  Produces an XML representation of this PVInfo according to the xml schemas.

Static Private Member Functions

- void **addStaticPV** (std::string name, std::string type, std::string cat, bool fac, std::string min, std::string max, bool visible=true)

  Adds a **PVDescription**(p. 357) *with the provided values.*

Static Private Attributes

- std::vector< **PVDescription** > **sStaticPV**

  Process variables known from the start.

- std::vector< **PVDescription** > **sSimulationDependentPV**

  Process varaibles that depends on the simulation (e.g. troop density etc.).

4.121.1 Detailed Description

Static class that holds information about the process variables that the server is capable of simulating.

Author:
Per Alexius

Date:
2007/01/24 13:13:23

4.121.2 Member Function Documentation

4.121.2.1 void PVInfo::addPV (std::string *n*, std::string *t*, std::string *c*, bool *f*, std::string *min*, std::string *max*) [static]

Adds a **PVDescription**(p. 357) with the provided values.

Parameters:

n The name of the PV.

t The type of the PV.

c The category of the PV.

f Flag for factions or not.

min The minimum value.

max The maximum value.

4.121.2.2 void PVInfo::addStaticPV (std::string *name*, std::string *type*, std::string *cat*, bool *fac*, std::string *min*, std::string *max*, bool *visible* = true) [static, private]

Adds a **PVDescription**(p. 357) with the provided values.

Parameters:

name The name of the PV.

type The type of the PV.

cat The category of the PV.

fac Flag for factions or not.

min The minimum value.

max The maximum value.

visible Flag indicating visibility to client.

4.121.2.3 ostream & PVInfo::toXML (std::ostream & *o*) [static]

Produces an XML representation of this PVInfo according to the xml schemas.

Parameters:

o The ostream to print to.

Returns:

The ostream with the XML representation written to it.

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

- PVInfo.h
- PVInfo.cpp

4.122 PVInitValue Class Reference

This class represents a ProcessVariableInitialValues xml object.

`#include <PVRegion.h>`

Public Member Functions

- **PVInitValue** (const DOMElement *n)
- ~**PVInitValue** ()
- eAllPV **pv** () const

  Accessor for the pv index.

- const std::vector< const **Reference** * > & **factions** () const

  Accessor for the factions vector.

- const std::vector< **PVRegion** * > & **regions** () const

  Accessor for region vector.

Private Attributes

- eAllPV **mPV**

  The pv index (as in the eAllPV enumeration).

- std::vector< const **Reference** * > **mFactions**

  References to the factions that the initialization should affect.

- std::vector< **PVRegion** * > **mRegions**

  The regions that the initialization should affect.

4.122.1 Detailed Description

This class represents a ProcessVariableInitialValues xml object.

Author:
Per Alexius

Date **Date**
2007/01/24 17:03:11

4.122.2 Constructor & Destructor Documentation

4.122.2.1 PVInitValue::PVInitValue (const DOMElement * *n*)

brief Constructor

Parameters:
n The DOMElement to create this object from.

4.122.2.2 PVInitValue::~PVInitValue ()

brief Destructor

4.122.3 Member Function Documentation

4.122.3.1 const std::vector<const Reference*>& PVInitValue::factions () const [inline]

Accessor for the factions vector.

Returns:
The factions vector.

4.122.3.2 eAllPV PVInitValue::pv () const [inline]

Accessor for the pv index.

Returns:
The pv index.

4.122.3.3 const std::vector<PVRegion*>& PVInitValue::regions () const [inline]

Accessor for region vector.

Returns:
The region vector.

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

- PVRegion.h
- PVRegion.cpp

4.123 PVInitValueSet Class Reference

This class represents a set of ProcessVariableInitialValues xml objects.

`#include <PVRegion.h>`

Public Member Functions

- **PVInitValueSet** (const DOMElement *n)
- ~**PVInitValueSet** ()
- const std::vector< **PVInitValue** * > & **initValues** () const

  Accessor for initial values vector.

Static Public Member Functions

- const **PVInitValueSet** * **currentSet** ()

  Accessor for the current set.

Private Attributes

- std::vector< **PVInitValue** * > **mInitValues**

  The vector containing the PVInitValues.

Static Private Attributes

- **PVInitValueSet** * **sCurrentSet**

  The current set used by the simulation.

4.123.1 Detailed Description

This class represents a set of ProcessVariableInitialValues xml objects.

Author:
Per Alexius

Date:
2007/01/24 17:03:11

4.123.2 Constructor & Destructor Documentation

4.123.2.1 PVInitValueSet::PVInitValueSet (const DOMElement * *n*)

brief Constructor

Parameters:
n The DOMElement to create this object from.

4.123.2.2 PVInitValueSet::~PVInitValueSet ()

brief Destructor

4.123.3 Member Function Documentation

4.123.3.1 const PVInitValueSet* PVInitValueSet::currentSet () [inline, static]

Accessor for the current set.

Returns:
The current set used by the simulation.

4.123.3.2 const std::vector<PVInitValue*>& PVInitValueSet::initValues () const [inline]

Accessor for initial values vector.

Returns:
The initial values vector.

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

- PVRegion.h
- PVRegion.cpp

4.124 PVModification Class Reference

This class represents a modification to a process variable.

`#include <PVArea.h>`

Public Types

- enum **ePVType** { **eSum**, **eMean**, **eUnknown** }

  Enumeration for aggregation types.

Public Member Functions

- **PVModification** (int pv, **EthnicFaction** &faction, double value)

  Constructor.

- int **pv** () const

  Accessor for the pv index.

- **EthnicFaction** & **faction** () const

  Accessor for the faction.

- double **value** () const

  Accessor for the value.

- int **type** () const

  Gets the aggregation type for the pv this modification refers to.

Static Public Member Functions

- int **type** (int t)

  Gets the aggregation type for the provided pv index.

Private Attributes

- int **mPV**

  The pv index (as in the eAllPV enumeration).

- **EthnicFaction** * **mFaction**

  The faction to modify.

- double **mValue**

  The pv value to use.

Static Private Attributes

- std::map< int, int > **sPVTypeMap**

  Contains a mapping between pv index (as in the eAllPV enumeration) to the type of aggregation that should be performed if several modifications are made to the same cell.

4.124.1 Detailed Description

This class represents a modification to a process variable.

Author:
Per Alexius

Date Date
2006/07/19 07:04:34

4.124.2 Constructor & Destructor Documentation

4.124.2.1 PVModification::PVModification (int *pv*, EthnicFaction & *faction*, double *value*) [inline]

Constructor.

Parameters:
pv The pv index (as in the eAllPV enumeration).

faction The faction to modify.

value The pv value.

4.124.3 Member Function Documentation

4.124.3.1 EthnicFaction& PVModification::faction () const [inline]

Accessor for the faction.

Returns:
The faction.

4.124.3.2 int PVModification::pv () const [inline]

Accessor for the pv index.

Returns:
The pv index.

4.124.3.3 int PVModification::type (int *t*) [inline, static]

Gets the aggregation type for the provided pv index.

Parameters:
t The pv index as in the eAllPV enumeration.

Returns:
The aggregation type.

4.124.3.4 int PVModification::type () const [inline]

Gets the aggregation type for the pv this modification refers to.

Returns:
The aggregation type.

4.124.3.5 double PVModification::value () const [inline]

Accessor for the value.

Returns:
The value.

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

- PVArea.h
- Grid.cpp

4.125 PVRegion Class Reference

This class represents a region and a pv value to set in that region.

`#include <PVRegion.h>`

Public Member Functions

- **PVRegion** (const DOMElement *n)
- ~**PVRegion** ()
- double **value** () const

  Accessor for the value.

- const **Shape** & **area** () const

  Accessor for the **Shape**(p. 412) *the modifications refer to.*

Private Attributes

- double **mValue**

  The pv value.

- const **Reference** * **mShapeRef**

  Reference(p. 378) *to a shape or null if it's a CreatedRegion.*

- const **Shape** * **mArea**

  The shape defining this region.

4.125.1 Detailed Description

This class represents a region and a pv value to set in that region.

Author:
Per Alexius

Date **Date**
2007/01/24 17:03:11

4.125.2 Constructor & Destructor Documentation

4.125.2.1 PVRegion::PVRegion (const DOMElement * *n*)

brief Constructor

Parameters:
n The DOMElement to create this object from.

4.125.2.2 PVRegion::~PVRegion ()

brief Destructor

4.125.3 Member Function Documentation

4.125.3.1 const Shape & PVRegion::area () const

Accessor for the **Shape**(p. 412) the modifications refer to.

Returns:
The **Shape**(p. 412).

4.125.3.2 double PVRegion::value () const [inline]

Accessor for the value.

Returns:
The value.

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

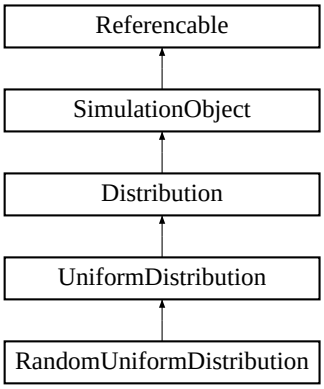
- PVRegion.h
- PVRegion.cpp

4.126 RandomUniformDistribution Class Reference

A random uniform distribution.

`#include <Distribution.h>`

Inheritance diagram for RandomUniformDistribution::



Public Member Functions

- **RandomUniformDistribution** (const **DataObject** &d)

  Creates a RandomUniformDistribution from the provided **DataObject**(p. 145).

- double **f** (double x) const

  Gets the value of the distribution at distance x.

4.126.1 Detailed Description

A random uniform distribution.

Author:
Per Alexius

Date:
2006/04/21 15:54:49

4.126.2 Constructor & Destructor Documentation

4.126.2.1 RandomUniformDistribution::RandomUniformDistribution (const DataObject & *d*) `[inline]`

Creates a RandomUniformDistribution from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.126.3 Member Function Documentation

4.126.3.1 double RandomUniformDistribution::f (double *x*) const [virtual]

Gets the value of the distribution at distance x.

Parameters:
x The distance.

Returns:
The value of the distribution at distance x.

Reimplemented from **UniformDistribution** (p. 543).

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

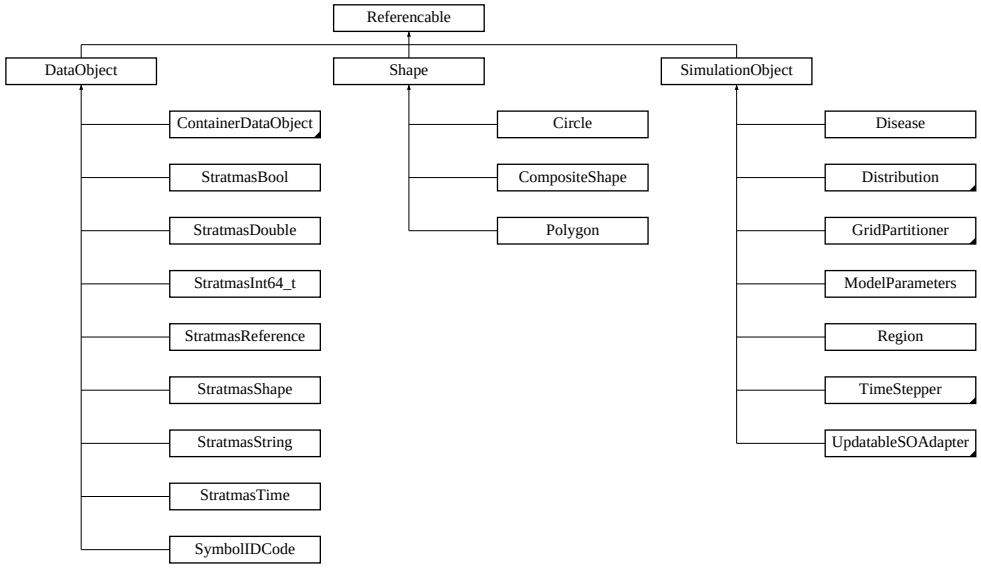
- Distribution.h
- Distribution.cpp

4.127 Referencable Class Reference

Superclass for all objects that could be pointed out by a **Reference**(p. 378).

`#include <Referencable.h>`

Inheritance diagram for Referencable::



Public Member Functions

- virtual ~**Referencable** ()

  Destructor.

- const **Reference** & **ref** () const

  Returns the **Reference**(p. 378) *to this Referenceable.*

Protected Member Functions

- **Referencable** ()

  Creates a Referencable refering to the null reference.

- **Referencable** (const **Reference** &scope, const DOMElement *n)

  Creates a Referencable in the specified scope, extracting the name from the provided DOMElement.

- **Referencable** (const **Reference** &ref)

  Creates a Referencable with **Reference**(p. 378) *ref.*

- **Referencable** (const **Referencable** &refable)

  Copy constructor.

Protected Attributes

- const **Reference** * **mReference**

  Pointer to the **Reference**(p. 378) *to this Object.*

4.127.1 Detailed Description

Superclass for all objects that could be pointed out by a **Reference**(p. 378).

Author:
Per Alexius

Date:
2006/05/24 12:32:10

4.127.2 Constructor & Destructor Documentation

4.127.2.1 Referencable::Referencable (const Reference & *scope*, const DOMElement * *n*) [protected]

Creates a Referencable in the specified scope, extracting the name from the provided DOMElement.

Parameters:
scope **Reference**(p. 378) to the scope of this Referenceable.

n Pointer to the DOMElement from which to extract the name of this Referencable.

4.127.2.2 Referencable::Referencable (const Reference & *ref*) [inline, protected]

Creates a Referencable with **Reference**(p. 378) ref.

Parameters:
ref The **Reference**(p. 378) to this Referenceable.

4.127.2.3 Referencable::Referencable (const Referencable & *refable*) [inline, protected]

Copy constructor.

Parameters:
refable The Referencable to copy.

4.127.3 Member Function Documentation

4.127.3.1 const Reference& Referencable::ref () const [inline]

Returns the **Reference**(p. 378) to this Referenceable.

Returns:
ref The **Reference**(p. 378) to this Referenceable.

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

- Referencable.h
- Referencable.cpp

4.128 Reference Class Reference

A Reference is used to point out an object somewhere in the Stratmas object hierarchy.

`#include <Reference.h>`

Public Member Functions

- const std::string & **name** () const

  Accessor for the name of this Reference.

- const **Reference** * **scope** () const

  Accessor for the scope of this Reference.

- bool **operator<** (const **Reference** &r) const

  Less than operator.

- bool **operator==** (const **Reference** &r) const

  Equality operator.

- bool **operator!=** (const **Reference** &r) const

  Inequality operator.

- std::ostream & **toXML** (std::ostream &o) const

  Produces an XML representation of this Reference according to the xml schemas.

- std::ostream & **toXML** (std::ostream &o, std::string indent) const

  Produces the XML representation of this object.

Static Public Member Functions

- const **Reference** & **root** ()

  Gets the root Reference.

- const **Reference** & **nullRef** ()

  Gets the null Reference.

- void **cleanUp** ()

  Deletes and frees memory for all existing References except the root and null references that are statically allocated.

- const **Reference** & **get** (const DOMElement *n)

  Gets the Reference specified in the provided DOMElement.

- const **Reference** & **get** (const **Reference** &scope, const std::string &name)

  Gets the Reference with the specified scope and name.

Private Member Functions

- **Reference** (const std::string &name)
  Creates a reference with the specified name. Only used when creating the root Reference.
- **Reference** (const **Reference** &scope, const std::string &name)
  Creates a reference with the specified name in the specified scope.
- **Reference** (const **Reference** &ref)
  Copy constructor. Mustn't be used.
- ~**Reference** ()
  Destructor.

Private Attributes

- const std::string **mName**
  The name of this Reference.
- const **Reference** * **mScope**
  The Reference making up the scope of this Reference.
- std::map< const std::string, const **Reference** * > **mChildren**
  Maps the name of a child of this Reference to the actual Reference of that child.

Static Private Attributes

- const **Reference sRoot**
  The global root Reference.
- const **Reference sNull**
  The global root Reference.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Reference** &r)
  For debugging purposes.

4.128.1 Detailed Description

A Reference is used to point out an object somewhere in the Stratmas object hierarchy.

Author:
Per Alexius

Date **Date**
2006/05/24 12:32:11

4.128.2 Constructor & Destructor Documentation

4.128.2.1 Reference::Reference (const std::string & *name*) [inline, private]

Creates a reference with the specified name. Only used when creating the root Reference.

Parameters:

name The name of the Reference to be created.

4.128.2.2 Reference::Reference (const Reference & *scope*, const std::string & *name*) [inline, private]

Creates a reference with the specified name in the specified scope.

Parameters:

scope The scope in which to create the Reference.

name The name of the Reference to be created.

4.128.2.3 Reference::Reference (const Reference & *ref*) [private]

Copy constructor. Mustn't be used.

Parameters:

ref The Reference that we aren't allowed to copy.

4.128.3 Member Function Documentation

4.128.3.1 const Reference & Reference::get (const Reference & *scope*, const std::string & *name*) [static]

Gets the Reference with the specified scope and name.

Parameters:

scope The scope of the Reference to get.

name The name of the Reference to get.

Returns:

The Reference with the specified scope and name.

4.128.3.2 const Reference & Reference::get (const DOMElement * *n*) [static]

Gets the Reference specified in the provided DOMElement.

Parameters:

n Pointer to the DOMElement to get the Reference from.

Returns:

The Reference extracted from the provided DOMElement.

4.128.3.3 const std::string& Reference::name () const [inline]

Accessor for the name of this Reference.

Returns:
The name of this Reference.

4.128.3.4 const Reference& Reference::nullRef () [inline, static]

Gets the null Reference.

Returns:
The null Reference.

4.128.3.5 bool Reference::operator!= (const Reference & *r*) const [inline]

Inequality operator.

Parameters:
r The Reference to compare to.

Returns:
True if this reference is not equal to r.

4.128.3.6 bool Reference::operator< (const Reference & *r*) const [inline]

Less than operator.

A Reference r1 is less than another reference r2 if:

r1's scope is less than r2's scope. A null scope is less than any other scope.

or

r1 and r2 have the same scope and r1's name is lexicographically less than r2's

Parameters:
r The Reference to compare to.

Returns:
True if this reference is less than r.

4.128.3.7 bool Reference::operator== (const Reference & *r*) const [inline]

Equality operator.

Two References are equal if they are located on the same memory address i.e. they are the same object.

Parameters:
r The Reference to compare to.

Returns:
True if this reference is equal to r.

4.128.3.8 const Reference& Reference::root () [inline, static]

Gets the root Reference.

Returns:
The root Reference.

4.128.3.9 const Reference* Reference::scope () const [inline]

Accessor for the scope of this Reference.

Returns:
The scope of this Reference.

4.128.3.10 ostream & Reference::toXML (std::ostream & *o*, std::string *indent*) const

Produces the XML representation of this object.

Parameters:
o The stream to which the object is written.

indent Intention string for readable output.

Returns:
The ostream with the XML representation written to it.

4.128.3.11 std::ostream& Reference::toXML (std::ostream & *o*) const [inline]

Produces an XML representation of this Reference according to the xml schemas.

Parameters:
o The ostream to print to.

Returns:
The ostream with the XML representation written to it.

4.128.4 Friends And Related Function Documentation

4.128.4.1 std::ostream& operator<< (std::ostream & *o*, const Reference & *r*) [friend]

For debugging purposes.

Parameters:
o The ostream to write to.

r The Reference to write.

Returns:
The ostream with the Reference written to it.

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

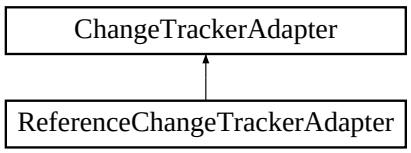
- Reference.h
- Reference.cpp

4.129 ReferenceChangeTrackerAdapter Class Reference

The ReferenceChangeTrackerAdapter keeps track of changes in **StratmasReference**(p. 476) objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for ReferenceChangeTrackerAdapter::



Public Member Functions

- **ReferenceChangeTrackerAdapter** (**StratmasReference** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 385) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **StratmasReference** & **mObject**

  The adapted **DataObject**(p. 145).

- const **Reference** ∗ **mLast**

  The last value written.

4.129.1 Detailed Description

The ReferenceChangeTrackerAdapter keeps track of changes in **StratmasReference**(p. 476) objects.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.129.2 Constructor & Destructor Documentation

4.129.2.1 ReferenceChangeTrackerAdapter::ReferenceChangeTrackerAdapter (StratmasReference & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.129.3 Member Function Documentation

4.129.3.1 bool ReferenceChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 385) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 385) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.129.3.2 ostream & ReferenceChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

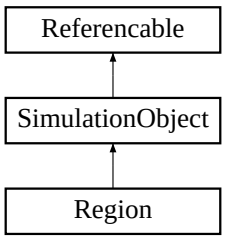
- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.130 Region Class Reference

This is the **SimulationObject**(p. 429) that corresponds to the Region type in the Stratmas xml schema.

`#include <Region.h>`

Inheritance diagram for Region::



Public Member Functions

- **Region** (const **DataObject** &d)

  Creates a Region from the provided **DataObject**(p. 145).

- virtual ~**Region** ()

  Destructor.

- const **Shape** & **area** () const

  Accessor for the area.

- const **CellGroup** & **cellGroup** () const
- void **prepareForSimulation** (const **Map** &map, **Grid** &grid, const **GridDataHandler** &gdh)

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- void **update** ()
- void **update** (const **Update** &u)

  Updates this object.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- double **pvrGet** (eRegionPV pvr) const
- void **pvrSet** (eRegionPV pvr, double value) const
- double **rp** (eRegionParameter param) const

Static Public Member Functions

- const char * **paramName** (eRegionParameter param)

Private Member Functions

- void **setDefault** ()

  Sets values to default.

- void **getDataFromDataObject** (const **DataObject** &d)

Private Attributes

- bool **mChanged**
- double * **mPVR**
- double * **mParam**
- const **Reference** * **mAreaRef**
- const **Shape** * **mArea**
- **CellGroup** * **mCellGroup**
- **Grid** * **mGrid**

Static Private Attributes

- std::map< std::string, eRegionParameter > **sNameToIndex**

4.130.1 Detailed Description

This is the **SimulationObject**(p. 429) that corresponds to the Region type in the Stratmas xml schema.

Author:
Per Alexius

Date **Date**
2006/10/02 16:05:14

4.130.2 Constructor & Destructor Documentation

4.130.2.1 Region::Region (const DataObject & *d*)

Creates a Region from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.130.3 Member Function Documentation

4.130.3.1 const Shape& Region::area () const [inline]

Accessor for the area.

Returns:
The area.

4.130.3.2 void Region::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.130.3.3 void Region::prepareForSimulation (const Map & *map*, Grid & *grid*, const GridDataHandler & *gdh*)

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Parameters:
map The map of the simulation.

4.130.3.4 void Region::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

4.130.3.5 void Region::update (const Update & *u*) [virtual]

Updates this object.

Parameters:
u The **Update**(p. 567) to update this object with.

Implements **SimulationObject** (p. 431).

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

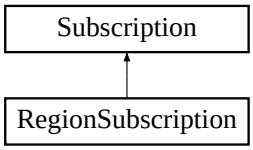
- Region.h
- Region.cpp

4.131 RegionSubscription Class Reference

RegionSubscription represents a subscription for a collection of cells.

`#include <Subscription.h>`

Inheritance diagram for RegionSubscription::



Public Member Functions

- **RegionSubscription** (DOMElement *n, **Buffer** &buf)

  Creates a subscription from a DOMElement, e.g. an xml representation.

- ~**RegionSubscription** ()

  Destructor.

- void **getSubscribedData** (std::ostream &o)

  Writes an XML representation of the subscribed data to the provided stream.

Private Member Functions

- void **printPV** (std::ostream &o, const char *name, const char *type, double value, **EthnicFaction** *faction=0)

Private Attributes

- std::list< **GridPos** > **mPositions**

  A list of grid positions this subscription covers.

- **CellGroup** * **mRegion**

  The group of cells this **Subscription**(p. 500) *refers to.*

- int **mResetCount**

  The reset count for the **Buffer**(p. 67).

4.131.1 Detailed Description

RegionSubscription represents a subscription for a collection of cells.

When the RegionSubscription is created it finds out which cells the specified **Region**(p. 386) overlaps and adds them to an AttributesGroup.

Author:
Per Alexius

Date:
2006/07/05 14:49:47

4.131.2 Constructor & Destructor Documentation

4.131.2.1 RegionSubscription::RegionSubscription (DOMElement * *n*, Buffer & *buf*)

Creates a subscription from a DOMElement, e.g. an xml representation.

Parameters:
n The DOMElement from which this subscription should be created.

buf The **Buffer**(p. 67) from which data should be fetched.

4.131.3 Member Function Documentation

4.131.3.1 void RegionSubscription::getSubscribedData (std::ostream & *o*) [virtual]

Writes an XML representation of the subscribed data to the provided stream.

Parameters:
o The stream to write to.

Implements **Subscription** (p. 501).

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

- Subscription.h
- Subscription.cpp

4.132 Registrator Class Reference

This class represents a registration to the dispatcher.

`#include <Registrator.h>`

Public Member Functions

- **Registrator** (std::string dHost, int dPort, std::string sHost, int sPort)

  Creates a new registrator.

- ~**Registrator** ()

  This destructor will leak the memory for dispatcherHost, and, possibly host.

- bool **registerServer** (int retries=4)

Private Member Functions

- bool **registerServerOnce** ()

Private Attributes

- std::string **dispatcherHost**

  The hostname of the dispatcher.

- int **dispatcherPort**

  The port of the dispatcher.

- std::string **host**

  The hostname of the server.

- int **port**

  The port of the server.

4.132.1 Detailed Description

This class represents a registration to the dispatcher.

Author:
Daniel Ahlin

Date: **Date**
2006/07/21 13:35:29

4.132.2 Constructor & Destructor Documentation

4.132.2.1 Registrator::Registrator (std::string *dHost*, int *dPort*, std::string *sHost*, int *sPort*)

Creates a new registrator.

Parameters:

dHost the name of the dispatcher.

dPort the port of the dispatcher.

sHost the name of the server.

sPort the port of the server.

4.132.3 Member Function Documentation

4.132.3.1 bool Registrator::registerServer (int *tries* = 4)

/brief Registers the server with the dispatcher.

4.132.3.2 bool Registrator::registerServerOnce () [private]

/brief Tries to register the server with the dispatcher.

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

- Registrator.h
- Registrator.cpp

4.133 Resetter< T > Class Template Reference

Convenience class that is used to handle resetting of vectors of SimulationObjects.

`#include <Resetter.h>`

Static Public Member Functions

- void **reset** (std::vector< T * > &simObjs, const std::vector< **DataObject** * > &dataObjs)

  Adds, removes and modifies the SimulationObjects in the simObjs vector based on the DataObjects in the dataObjs vector so that the SimulationObjects looks just like they would have if they were created from the provided DataObjects.

4.133.1 Detailed Description

template<class T> class Resetter< T >

Convenience class that is used to handle resetting of vectors of SimulationObjects.

This class is used when the client resets a **Simulation**(p. 424) in order to restore the original set of SimulationObjects in a vector based on the original corresponding vector of DataObjects.

Author:
Per Alexius

Date:
2006/07/10 08:58:35

4.133.2 Member Function Documentation

4.133.2.1 template<class T> void Resetter< T >::reset (std::vector< T * > & *simObjs*, const std::vector< DataObject * > & *dataObjs*) [static]

Adds, removes and modifies the SimulationObjects in the simObjs vector based on the DataObjects in the dataObjs vector so that the SimulationObjects looks just like they would have if they were created from the provided DataObjects.

Parameters:

simObjs The vector containing the SimulationObjects to be reset. On exit this vector contains the SimulationObjects as they would have looked like if they were created from the DataObjects in the dataObjs vector.

dataObjs The vector of DataObjects to reset the SimulationObjects with.

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

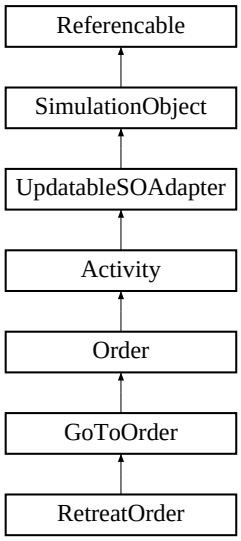
- Resetter.h

4.134 RetreatOrder Class Reference

The RetreatOrder. This order only exists on the server side.

`#include <Activity.h>`

Inheritance diagram for RetreatOrder::



Public Types

- enum **eRetreatState** { **eRetreat**, **eReturn** }

  Enumeration for RetreatOrder state. The eRetreat state means that the unit is retreating x km in a random direction. The eReturn state means that the unit has retreated x km in a random direction and is now on its way towards its closest superior (untouchable and with half ordinary velocity).

Public Member Functions

- **RetreatOrder** (const **Shape** &location)

  Constructor used internally by the server.

- void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

- int **state** () const

  Accessor for the state.

- **Unit** * **findClosestParentWithSameFaction** (**Unit** &u)

Private Attributes

- int **mState**

The state of the RetreatOrder.

4.134.1 Detailed Description

The RetreatOrder. This order only exists on the server side.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.134.2 Constructor & Destructor Documentation

4.134.2.1 RetreatOrder::RetreatOrder (const Shape & *location*) [inline]

Constructor used internally by the server.

Parameters:
location The location to retreat to.

4.134.3 Member Function Documentation

4.134.3.1 void RetreatOrder::perform (Element * *e*, double *fraction* = 1.0) [virtual]

Performs this **Activity**(p. 21).

Parameters:
e The **Element**(p. 180) that should perform this **Activity**(p. 21).

fraction The fraction of the performers total capacity that this activity is performed with.

Reimplemented from **GoToOrder** (p. 226).

4.134.3.2 int RetreatOrder::state () const [inline]

Accessor for the state.

Returns:
The state.

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

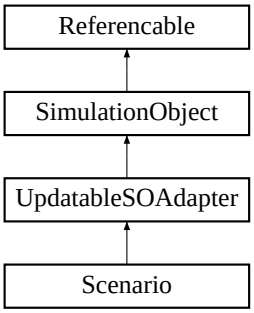
- Activity.h
- Activity.cpp

4.135 Scenario Class Reference

This class represents the simulation instance of a Scenario.

`#include <Scenario.h>`

Inheritance diagram for Scenario::



Public Member Functions

- **Scenario** (const **DataObject** &d)

  Creates a Scenario simulation object from the provided DataObjec.

- ~**Scenario** ()

  Destructor.

- **GridDataHandler** * **takeOverGridDataHandler** () const
- void **prepareForSimulation** (const **GridPartitioner** &g, const **ModelParameters** &m, **Time** startTime)

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- void **step** (**Time** currentTime)

  Advances the simulation to the specified time.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- void **createCityRegionMapping** ()

  Helper function used to find out which region contains which cities. Not used during simulation but only when converting old scenarios (taclan and other necessary files) files to new taclan2 files.

Private Attributes

- **TimeStepper** * **mTimeStepper**

  The **TimeStepper**(p. 522).

- **Map** * **mMap**

  The **Map**(p. 292) *of the scenario.*

- **Grid** * **mGrid**

  The simulation grid.

- **CombatGrid** * **mCombatGrid**

  The combat grid.

- int **mNumEthnicFactions**

  The number of ethnic factions in the scenario.

- std::vector< **Faction** * > **mFactions**

  A vector with all Factions.

- std::vector< **City** * > **mCities**

  A vector with all Cities.

- std::vector< **Unit** * > **mForces**

  A vector with the roots of the force hierarchies.

- std::vector< **AgencyTeam** * > **mAgencyTeams**

  A vector with all agency teams.

- std::vector< **Agency** * > **mAgencies**

  A vector with all agencies.

- std::vector< **Activity** * > **mActivities**

  A vector with all Activities.

- std::vector< **Region** * > **mRegions**

  A vector with all Regions.

- **Disease** * **mDisease**

  An pointer to the disease object.

- **ModelParameters** * **mModelParameters**

  An object containing various model parameters.

- **AgencyFactory * mAgencyFactory**
  The factory used to create Agencies.
- **Time mNextGridUpdate**
  Time(p. 516) *when the* **Grid**(p. 227) *should be updated the next time.*
- **Time mCurrentTime**
  The time for the last call to **step()**(p. 400).
- double **mHDI**
  The HDI parameter.
- double **mUnemployment**
  The unemployment parameter.
- **GridDataHandler * mGridDataHandler**

4.135.1 Detailed Description

This class represents the simulation instance of a Scenario.

Author:
Per Alexius

Date Date
2006/10/02 16:05:15

4.135.2 Constructor & Destructor Documentation

4.135.2.1 Scenario::Scenario (const DataObject & *d*)

Creates a Scenario simulation object from the provided DataObjec.

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.135.3 Member Function Documentation

4.135.3.1 void Scenario::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:
toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.
initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 564).

4.135.3.2 void Scenario::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.135.3.3 void Scenario::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.135.3.4 void Scenario::prepareForSimulation (const GridPartitioner & *g*, const ModelParameters & *m*, Time *startTime*)

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Parameters:
g The GridPartitoner to use when creating the grid.
m The **ModelParameters**(p. 301) object.
startTime The start time of the simulation.

4.135.3.5 void Scenario::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:
toRemove The **Reference**(p. 378) to the object to remove.
initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.135.3.6 void Scenario::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

4.135.3.7 void Scenario::step (Time *currentTime*)

Advances the simulation to the specified time.

Parameters:

currentTime The current simulation time.

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

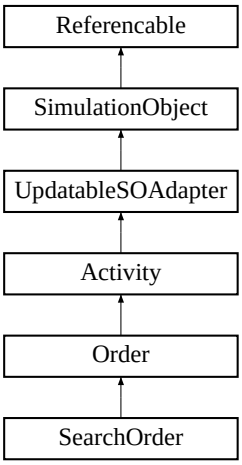
- Scenario.h
- Scenario.cpp

4.136 SearchOrder Class Reference

The SearchOrder for searching for TerroristAttacking units. This order only exists on the server side.

`#include <Activity.h>`

Inheritance diagram for SearchOrder::



Public Member Functions

- **SearchOrder** (**Time** end)

  Constructor.

- bool **isActive** (**Time** t)

  Checks if this activity is active at time t.

- void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

- double **combatFactor** () const

  Accessor for the combat factor.

Protected Attributes

- **Time mEnd**

  The end time.

4.136.1 Detailed Description

The SearchOrder for searching for TerroristAttacking units. This order only exists on the server side.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.136.2 Constructor & Destructor Documentation

4.136.2.1 SearchOrder::SearchOrder (Time *end*) [inline]

Constructor.

Parameters:
end The end time of the order.

4.136.3 Member Function Documentation

4.136.3.1 double SearchOrder::combatFactor () const [inline, virtual]

Accessor for the combat factor.

Returns:
The combat factor.

Implements **Order** (p. 310).

4.136.3.2 bool SearchOrder::isActive (Time *t*) [inline, virtual]

Checks if this activity is active at time t.

Parameters:
t The time for which to check.

Returns:
True if this activity is active at the specified time.

Reimplemented from **Order** (p. 310).

4.136.3.3 void SearchOrder::perform (Element * *e*, double *fraction* = 1.0) [inline, virtual]

Performs this **Activity**(p. 21).

Parameters:
e The **Element**(p. 180) that should perform this **Activity**(p. 21).
fraction The fraction of the performers total capacity that this activity is performed with.

Implements **Activity** (p. 23).

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

- Activity.h

4.137 Server Class Reference

This class represents the server.

`#include <Server.h>`

Public Member Functions

- **Server** (int port, const std::string &host, **ClientValidator** *clientValidator)

  Creates a server object.

- ~**Server** ()

  Destructor.

- void **start** ()

  Starts the server.

- void **notifyClosure** (int64_t id)

  Notifies the server about a **Session**(p. 408) *that was ended.*

- bool **hasActiveClient** () const

  Checks if the Server currently has an active **Session**(p. 408)*, i.e. if there is a client connected that is active.*

- const std::map< int64_t, **Session** * > & **sessions** () const

  Accessor for the map of Sessions.

Static Public Member Functions

- void **sigpipe_handle** (int sig)
- void * **dispatcherThreadMain** (void *data)

  Main function of the dispatcher thread.

- bool **bigEndian** ()

  Returns endian of current architecture.

- void **handleTemporarySession** (**Server** &server, **StratmasSocket** &sock)

  Handles temporary Sessions.

Private Attributes

- **StratmasServerSocket** * **mSocket**

  The socket used to receive connections .

- int **mNumSessions**

  Current number of sessions.

- int64_t **mIdCount**

  Keeps track of which id to give to the next **Session**(p. 408).

- std::map< int64_t, **Session** * > **mSessions**

  Mapping session id to **Session**(p. 408) *object.*

- **Buffer** * **mBuf**

  Pointer to the **Buffer**(p. 67) *object.*

- **Engine** * **mEng**

  Pointer to the **Engine**(p. 189) *object.*

- int64_t **mActiveId**

  Id of the avtive **Session**(p. 408).

- **TSQueue**< **StratmasSocket** * > **mConQ**

  Queue used to communicate with the dispatcher thread.

- **ClientValidator** * **mClientValidator**

  Used to validate incomming connections.

Static Private Attributes

- bool **sBigEndian** = (!(*(char*)&one))

  Set to true if the current architecture is big endian.

4.137.1 Detailed Description

This class represents the server.

Author:
Per Alexius

Date:
2006/07/24 10:14:35

4.137.2 Constructor & Destructor Documentation

4.137.2.1 Server::Server (int *port*, const std::string & *host*, ClientValidator * *clientValidator*)

Creates a server object.

Parameters:

port The port to listen to connections on.

host The interface to listen to connections on.

client Explicitly defined ip from which to allow connections or NULL if no such client was specified.

validIPFile Name of explicitly specified validIPFile or NULL if no such file was specified.

ipValidation Set to true if ip validation should be applied, false otherwise.

4.137.3 Member Function Documentation

4.137.3.1 bool Server::bigEndian () [inline, static]

Returns endian of current architecture.

Returns:
Endian of current architecture.

4.137.3.2 void * Server::dispatcherThreadMain (void * *data*) [static]

Main function of the dispatcher thread.

The dispatcher waits for a **StratmasSocket**(p. 485) (containing data about a connection) to be enqueued by the Server main thread. When this happens, the dispatcher creates a thread for handling the connection and gives that thread access to the correct **Session**(p. 408) object. The dispatcher keeps doing this until it dequeues a NULL pointer that indicates that it is ok to quit.

Parameters:
data A Server object.

Returns:
NULL if everything is ok.

4.137.3.3 void Server::handleTemporarySession (Server & *server*, StratmasSocket & *sock*) [static]

Handles temporary Sessions.

Used for LoadQueryMessages.

Parameters:
server The server object.

sock The **StratmasSocket**(p. 485) that the temporary session uses.

4.137.3.4 bool Server::hasActiveClient () const [inline]

Checks if the Server currently has an active **Session**(p. 408), i.e. if there is a client connected that is active.

Returns:
True if there is an active client.

4.137.3.5 void Server::notifyClosure (int64_t *id*)

Notifies the server about a **Session**(p. 408) that was ended.

Parameters:

id The id of the **Session**(p. 408) that was ended.

4.137.3.6 const std::map<int64_t, Session*>& Server::sessions () const [inline]

Accessor for the map of Sessions.

Returns:

The map of Sessions.

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

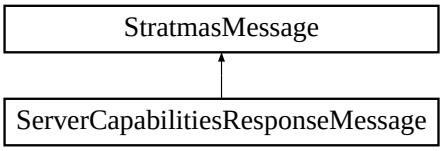
- Server.h
- Server.cpp

4.138 ServerCapabilitiesResponseMessage Class Reference

Class representing the ServerCapabilitiesResponseMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for ServerCapabilitiesResponseMessage::



Public Member Functions

- **ServerCapabilitiesResponseMessage** ()
  Constructor.
- void **toXML** (std::ostream &o) const
  Produces the XML representation of this message.

4.138.1 Detailed Description

Class representing the ServerCapabilitiesResponseMessage.

Author:
Per Alexius

Date:
2006/03/06 14:23:12

4.138.2 Member Function Documentation

4.138.2.1 void ServerCapabilitiesResponseMessage::toXML (std::ostream & *o*) const [virtual]

Produces the XML representation of this message.

Parameters:
o The stream to which the message is written

Implements **StratmasMessage** (p. 473).

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

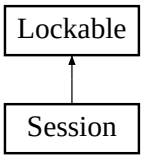
- StratmasMessage.h
- StratmasMessage.cpp

4.139 Session Class Reference

Class that handles a session between the **Server**(p. 403) and a client.

`#include <Session.h>`

Inheritance diagram for Session::



Public Member Functions

- **Session** (**Server** &parent, **Engine** &e, **Buffer** &b, bool isMaster, **StratmasSocket** *s)
  Creates a Session object.
- ~**Session** ()
  Destructor.
- void **closeSession** ()
  Closes this session.
- void **handleStratmasMessage** (const std::string &xml, std::string &response)
  Takes a **StratmasMessage**(p. 472)*, handles it and produces a response.*
- void **handleInitialization** ()
  Handles initialization.
- void **start** ()
  Starts the Session.
- bool **setSocket** (**StratmasSocket** *s)
  Sets the socket this Session should use for communication.
- int64_t **id** () const
  Accessors for the id.
- bool **isActive** () const
  Accessors for the active flag.
- std::string **simulationName** () const
  Returns the name of the simulation this session handles.

Static Public Member Functions

- void * **staticStart** (void *instance)
  Main method of threads handling a Session.

Private Attributes

- **Server** & **mServer**
  The **Server**(p. 403) *object that created this Session.*
- **Engine** & **mEng**
  Reference(p. 378) *to the* **Engine**(p. 189) *object.*
- **Buffer** & **mBuf**
  Reference(p. 378) *to the* **Buffer**(p. 67) *object.*
- int64_t **mId**
  Id of this Session.
- bool **mActive**
  Indicates if this Session is with is an active client or not.
- bool **mBigEndian**
  Endian of the architecture of the client.
- bool **mDisconnect**
  Set to true when we should disconnect.
- **StratmasSocket** * **mSocket**
  The socket over which to communicate.
- **XMLHandler** * **mXMLHandler**
  The **XMLHandler**(p. 582) *Object that handles parsing of xml messages.*
- **Time mLastSentTime**
  Keeps the last simulation time for which data was sent to the client.
- **TSQueue**< **EngineStatusObject** > **mQueue**
  Queue used by the **Engine**(p. 189) *thread to communicate with us.*
- int **mBufferResetCount**
  Keep track of the buffer's reset count.
- **ContainerChangeTrackerAdapter** * **mChangeTracker**
  The **ChangeTrackerAdapter**(p. 82) *that keeps track of changes in the* **Simulation**(p. 424).
- bool **mRegisteredForUpdates**
  Status flag indicating if the client has registered for updates.

4.139.1 Detailed Description

Class that handles a session between the **Server**(p. 403) and a client.

When the **Server**(p. 403) receives a ConnectMessage from a client a Session is created to handle further connumication from that client regarding that specific Session. The Session then lives until that client sends a DisconnectMessage.

Author:
Per Alexius

Date Date
2006/07/24 10:14:36

4.139.2 Constructor & Destructor Documentation

4.139.2.1 Session::Session (Server & *parent*, Engine & *e*, Buffer & *b*, bool *isActive*, StratmasSocket * *s*)

Creates a Session object.

Parameters:

parent The **Server**(p. 403) that created this object.

e A reference to the **Engine**(p. 189) object.

b A reference to the **Buffer**(p. 67) object.

isActive Indicates if this client is active ro not.

s A pointer to the **Socket**(p. 432) to use for communication.

4.139.3 Member Function Documentation

4.139.3.1 void Session::closeSession ()

Closes this session.

Sets the disconnect flag to true and if this is an active client session - tells the **Engine**(p. 189) to end the current **Scenario**(p. 396).

4.139.3.2 void Session::handleStratmasMessage (const std::string & *xml*, std::string & *response*)

Takes a **StratmasMessage**(p. 472), handles it and produces a response.

Parameters:

xml The message to handle.

response The message to send as response.

4.139.3.3 int64_t Session::id () const [inline]

Accessors for the id.

Returns:
The id of this Session.

4.139.3.4 bool Session::isActive () const [inline]

Accessors for the active flag.

Returns:
The state of the active flag.

4.139.3.5 bool Session::setSocket (StratmasSocket * *s*)

Sets the socket this Session should use for communication.

Parameters:
s A pointer to the socket to use for communication.

4.139.3.6 string Session::simulationName () const

Returns the name of the simulation this session handles.

Returns:
The name of the simulation this session handles.

4.139.3.7 void * Session::staticStart (void * *instance*) [static]

Main method of threads handling a Session.

Parameters:
instance An instance of the Session class that this thread should handle.

Returns:
NULL if everything is ok.

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

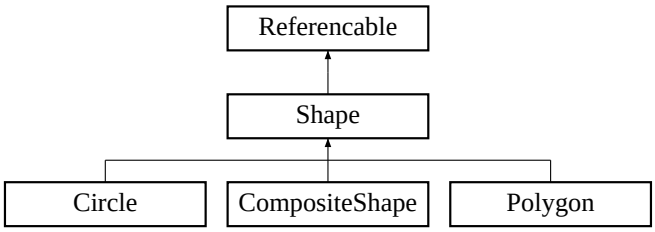
- Session.h
- Session.cpp

4.140 Shape Class Reference

An abstract base class for all Shapes.

`#include <Shape.h>`

Inheritance diagram for Shape::



Public Member Functions

- **Shape** ()
  Default constructor.
- **Shape** (const **Reference** &ref)
  Creates a Shape with the specified **Reference**(p. 378).
- virtual ~**Shape** ()
  Destructor.
- void **touch** ()
  Make this shape believe that it has been modified.
- bool **projected** () const
  Accessor for the projected flag.
- int **changes** () const
  Accessor for the number of changes.
- virtual void **toProj** ()
  Projects this Shape using the current projection.
- virtual void **toProj** (const **Projection** &proj)=0
  Projects this Shape using the specified **Projection**(p. 348).
- virtual void **toCoord** ()
  Transforms this Shape to lat lng coordinate using the current projection.
- virtual void **toCoord** (const **Projection** &proj)=0
  Transforms this Shape to lat lng coordinate using the provided projection.
- virtual void **cells** (const **BasicGrid** &g, std::list< **GridPos** > &outCells) const =0
  Returns a list containing pointers to all cells covered by this Shape.

- virtual **LatLng cenCoord** () const =0

  Returns the center coordinate in lat lng of this Shape.

- virtual **ProjCoord cenProj** () const =0

  Returns the center coordinate in projection space of this Shape.

- virtual void **boundingBox** (double &t, double &l, double &b, double &r) const =0

  Gets the bounding box of this Shape.

- virtual double **area** () const =0

  Returns the area of this Shape.

- virtual void **move** (double dx, double dy)=0

  Moves this Shape relative to itself.

- virtual void **move** (**LatLng** newPos)=0

  Moves this Shape to a new position.

- virtual **Shape** * **clone** () const =0

  Creates a deep copy of this Shape.

- virtual const std::string **type** () const =0

  Returns the stratmas protocol type of this shape.

- std::ostream & **toXML** (std::ostream &o) const

  Writes an XML representation of this object to the provided stream.

- virtual std::ostream & **toXML** (std::ostream &o, std::string indent) const =0

  Writes an XML representation of this object to the provided stream with nice indentation.

- virtual std::ostream & **cellsToXML** (const **BasicGrid** &grid, bool swapEndian, std::ostream &o) const

  Writes an XML representation of the cells covered by this shape to the provided stream.

Protected Attributes

- bool **mProjected**

  Indicates whether this shape is stored in projected coordinates or not.

- int **mChanges**

  Keeps track of the number of times this Shape has been changed.

4.140.1 Detailed Description

An abstract base class for all Shapes.

For more documentation of the various functions - see the subclasses that implement them.

Author:
Per Alexius

Date Date
2006/07/19 07:04:39

4.140.2 Constructor & Destructor Documentation

4.140.2.1 Shape::Shape (const Reference & *ref*)

Creates a Shape with the specified **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) this Shape should be given.

4.140.3 Member Function Documentation

4.140.3.1 virtual double Shape::area () const [pure virtual]

Returns the area of this Shape.

Returns:
The area of this Shape.

Implemented in **Circle** (p. 88), **Polygon** (p. 329), and **CompositeShape** (p. 115).

4.140.3.2 virtual void Shape::boundingBox (double & *t*, double & *l*, double & *b*, double & *r*) const [pure virtual]

Gets the bounding box of this Shape.

Parameters:
t Top coordinate of this Shape's boundingbox.
l Left coordinate of this Shape's boundingbox.
b Bottom coordinate of this Shape's boundingbox.
r Right coordinate of this Shape's boundingbox.

Implemented in **Circle** (p. 88), **Polygon** (p. 329), and **CompositeShape** (p. 115).

4.140.3.3 virtual void Shape::cells (const BasicGrid & *g*, std::list< GridPos > & *outCells*) const [pure virtual]

Returns a list containing pointers to all cells covered by this Shape.

Parameters:

g A reference to the **Grid**(p. 227).

outCells A list that on return contains pointers to all cells covered by this Shape.

Implemented in **Circle** (p. 88), **Polygon** (p. 329), and **CompositeShape** (p. 116).

4.140.3.4 ostream & Shape::cellsToXML (const BasicGrid & *grid*, bool *swapEndian*, std::ostream & *o*) const [virtual]

Writes an XML representation of the cells covered by this shape to the provided stream.

This method is used when telling the client which cells a certain region covers.

Parameters:

grid A **Reference**(p. 378) to the **Grid**(p. 227).

swapEndian Indicates if we have to change byte order in the produced data.

o The stream to write to.

Returns:

The stream with the xml representation written to it.

4.140.3.5 virtual LatLng Shape::cenCoord () const [pure virtual]

Returns the center coordinate in lat lng of this Shape.

Returns:

The center coordinate of this Shape.

Implemented in **Circle** (p. 89), **Polygon** (p. 330), and **CompositeShape** (p. 116).

4.140.3.6 virtual ProjCoord Shape::cenProj () const [pure virtual]

Returns the center coordinate in projection space of this Shape.

Returns:

The center coordinate of this Shape.

Implemented in **Circle** (p. 89), **Polygon** (p. 330), and **CompositeShape** (p. 116).

4.140.3.7 int Shape::changes () const [inline]

Accessor for the number of changes.

Returns:

The number of changes.

4.140.3.8 virtual Shape* Shape::clone () const [pure virtual]

Creates a deep copy of this Shape.

Returns:
A newly allocated copy of this Shape.

Implemented in **Circle** (p. 89), **Polygon** (p. 330), and **CompositeShape** (p. 116).

4.140.3.9 virtual void Shape::move (LatLng *newPos*) [pure virtual]

Moves this Shape to a new position.

Parameters:
newPos The position to move to.

Implemented in **Circle** (p. 89), **Polygon** (p. 331), and **CompositeShape** (p. 117).

4.140.3.10 virtual void Shape::move (double *dx*, double *dy*) [pure virtual]

Moves this Shape relative to itself.

Parameters:
dx The movement in x-direction in degrees longitude
dy The movement in y-direction in degrees latitude

Implemented in **Circle** (p. 89), **Polygon** (p. 331), and **CompositeShape** (p. 117).

4.140.3.11 bool Shape::projected () const [inline]

Accessor for the projected flag.

Returns:
The status of the projected flag.

4.140.3.12 virtual void Shape::toCoord (const Projection & *proj*) [pure virtual]

Transforms this Shape to lat lng coordinate using the provided projection.

Parameters:
proj The projection to use.

Implemented in **Circle** (p. 90), **Polygon** (p. 331), and **CompositeShape** (p. 118).

4.140.3.13 virtual void Shape::toProj (const Projection & *proj*) [pure virtual]

Projects this Shape using the specified **Projection**(p. 348).

Parameters:
proj The projection to use.

Implemented in **Circle** (p. 90), **Polygon** (p. 331), and **CompositeShape** (p. 118).

4.140.3.14 virtual std::ostream& Shape::toXML (std::ostream & *o*, std::string *indent*) const [pure virtual]

Writes an XML representation of this object to the provided stream with nice indentation.

Parameters:

o The stream to write to.

indent Indentation string.

Returns:

The stream with the xml representation written to it.

Implemented in **Circle** (p. 90), **Polygon** (p. 331), and **CompositeShape** (p. 118).

4.140.3.15 std::ostream& Shape::toXML (std::ostream & *o*) const [inline]

Writes an XML representation of this object to the provided stream.

Parameters:

o The stream to write to.

Returns:

The stream with the xml representation written to it.

4.140.3.16 virtual const std::string Shape::type () const [pure virtual]

Returns the stratmas protocol type of this shape.

Returns:

The stratmas protocol type of this shape.

Implemented in **Circle** (p. 90), **Polygon** (p. 332), and **CompositeShape** (p. 118).

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

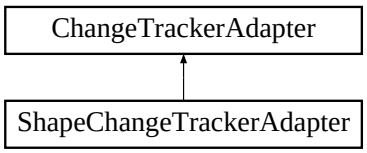
- Shape.h
- Shape.cpp

4.141 ShapeChangeTrackerAdapter Class Reference

The ShapeChangeTrackerAdapter keeps track of changes in **StratmasShape**(p. 481) objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for ShapeChangeTrackerAdapter::



Public Member Functions

- **ShapeChangeTrackerAdapter** (**StratmasShape** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 419) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **StratmasShape** & **mObject**
- int **mLast**

  The change count of the last **Shape**(p. 412) *written.*

4.141.1 Detailed Description

The ShapeChangeTrackerAdapter keeps track of changes in **StratmasShape**(p. 481) objects.

Author:
Per Alexius

Date **Date**
2006/03/02 17:06:51

4.141.2 Constructor & Destructor Documentation

4.141.2.1 ShapeChangeTrackerAdapter::ShapeChangeTrackerAdapter (StratmasShape & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.141.3 Member Function Documentation

4.141.3.1 bool ShapeChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 419) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 419) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.141.3.2 ostream & ShapeChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

4.141.4 Member Data Documentation

4.141.4.1 StratmasShape& ShapeChangeTrackerAdapter::mObject [private]

The adapted **DataObject**(p. 145).

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

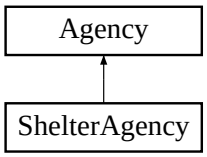
- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.142 ShelterAgency Class Reference

Class containing functionality for controlling ShelterAgencyTeams.

`#include <Agency.h>`

Inheritance diagram for ShelterAgency::



Public Member Functions

- **ShelterAgency** (const std::vector< **AgencyTeam** * > &teams, **Grid** &g)

  Constructor.

- void **act** (**Time** now)

  Determine actions of the ShelterAgencyTeams.

Private Member Functions

- bool **severeProblem** ()

  Determines if we have a severe problem. If we do - then the weights for the clustering algorithm is set.

Private Attributes

- int **mOperationalTeams**

  The number of operational teams.

4.142.1 Detailed Description

Class containing functionality for controlling ShelterAgencyTeams.

Author:
Per Alexius

Date:
2006/10/02 16:01:25

4.142.2 Constructor & Destructor Documentation

4.142.2.1 ShelterAgency::ShelterAgency (const std::vector< AgencyTeam * > & *teams*, Grid & *g*) [inline]

Constructor.

Parameters:

teams A vector containing this Agency's teams.

g A reference to the **Grid**(p. 227).

4.142.3 Member Function Documentation

4.142.3.1 void ShelterAgency::act (Time *now*) [virtual]

Determine actions of the ShelterAgencyTeams.

Find out if we have severe problems with displaced unsheltered people. If so - order teams to start building camps at the locations for the problems.

Parameters:

now The current simulation time.

Reimplemented from **Agency** (p. 27).

4.142.3.2 bool ShelterAgency::severeProblem () [private, virtual]

Determines if we have a severe problem. If we do - then the weights for the clustering algorithm is set.

Returns:

True if we have a severe displaced unsheltered people problem, false otherwise.

Implements **Agency** (p. 28).

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

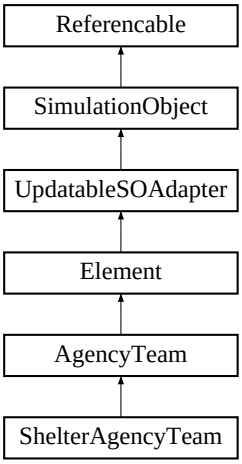
- Agency.h
- Agency.cpp

4.143 ShelterAgencyTeam Class Reference

Class representing a ShelterAgencyTeam.

`#include <AgencyTeam.h>`

Inheritance diagram for ShelterAgencyTeam::



Public Member Functions

- **ShelterAgencyTeam** (const **DataObject** &d)
  Constructor that creates an **AgencyTeam**(p. 32) *from the provided* **DataObject**(p. 145).
- void **act** (**Time** now)
  Performs this team's actions.

4.143.1 Detailed Description

Class representing a ShelterAgencyTeam.

Author:
Per Alexius

Date:
2006/10/10 09:35:59

4.143.2 Constructor & Destructor Documentation

4.143.2.1 ShelterAgencyTeam::ShelterAgencyTeam (const DataObject & *d*) [inline]

Constructor that creates an **AgencyTeam**(p. 32) from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.143.3 Member Function Documentation

4.143.3.1 void ShelterAgencyTeam::act (Time *now*) [virtual]

Performs this team's actions.

A ShelterAgencyTeam's action pattern is as follows: When it becomes operational, it builds a **Camp**(p. 74) at its current location. For all following days the team moves all of the displaced, unsheltered people in the cell where the camp is located into the camp untilthe team's - and thereby the camp's - capcaity limit is met.

Parameters:

now The current simulation time.

Implements **AgencyTeam** (p. 35).

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

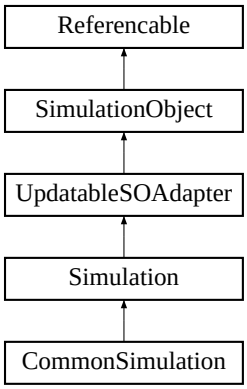
- AgencyTeam.h
- AgencyTeam.cpp

4.144 Simulation Class Reference

A base class for all Simulations.

`#include <Simulation.h>`

Inheritance diagram for Simulation::



Public Member Functions

- virtual ~**Simulation** ()

  Destructor.

- **GridDataHandler** * **takeOverGridDataHandler** () const
- void **prepareForSimulation** ()

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- **Time step** ()

  Advances the simulation one timestep.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Static Public Member Functions

- **Time timestep** ()

  Accessor for the size of the current time step.

- **Time simulationTime** ()

  Accessor for the current simultion time.

- double **fractionOfDay** ()

  Returns the fraction of a day that the current time step constitutes.

Protected Member Functions

- **Simulation** (const **DataObject** &d)

  The **ParameterGroup**(p. 314) *object.*

Private Attributes

- **TimeStepper** * **mTimeStepper**

  The **TimeStepper**(p. 522) *to use when stepping this Simulation.*

- **GridPartitioner** * **mGridPartitioner**

  The **GridPartitioner**(p. 254) *to uses when partitioning the* **Grid**(p. 227).

- **Scenario** * **mScenario**

  The **Scenario**(p. 396) *object.*

- **Time mStartTime**

  The start time of the Simulation.

- unsigned long **mRandomSeed**

  The random seed.

- **ModelParameters** * **mModelParameters**

  The **ModelParameters**(p. 301) *object.*

Static Private Attributes

- **Time sTimestep**

  The current timestep.

- **Time sSimTime**

  The current simulation time.

4.144.1 Detailed Description

A base class for all Simulations.

Author:
Per Alexius

Date:
2006/10/02 16:05:16

4.144.2 Constructor & Destructor Documentation

4.144.2.1 Simulation::Simulation (const DataObject & *d*) [protected]

The **ParameterGroup**(p. 314) object.

Parameters:
d The data object to create this object from.

4.144.3 Member Function Documentation

4.144.3.1 void Simulation::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:
toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.
initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 564).

4.144.3.2 void Simulation::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.144.3.3 double Simulation::fractionOfDay () [inline, static]

Returns the fraction of a day that the current time step constitutes.

Returns:
The fraction of a day that the current time step constitutes.

4.144.3.4 void Simulation::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.144.3.5 void Simulation::prepareForSimulation ()

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

4.144.3.6 void Simulation::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:
toRemove The **Reference**(p. 378) to the object to remove.
initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.144.3.7 void Simulation::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

4.144.3.8 Time Simulation::simulationTime () [inline, static]

Accessor for the current simultion time.

Returns:
The current simultion time.

4.144.3.9 Time Simulation::step ()

Advances the simulation one timestep.

Returns:
The simulation time after the step was taken.

4.144.3.10 Time Simulation::timestep () [inline, static]

Accessor for the size of the current time step.

Returns:
The size of the current time step.

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

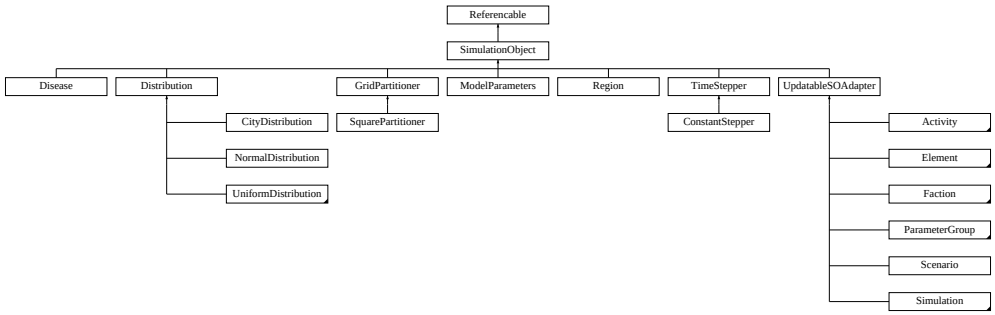
- Simulation.h
- Simulation.cpp

4.145 SimulationObject Class Reference

An abstract base class for all SimulationObjects.

`#include <SimulationObject.h>`

Inheritance diagram for SimulationObject::



Public Member Functions

- virtual ~**SimulationObject** ()

  Destructor.

- virtual void **update** (const **Update** &u)=0

  Updates this object.

- virtual void **extract** (**Buffer** &b) const =0

  Extracts data from this object to the **Buffer**(p. 67).

- virtual void **reset** (const **DataObject** &d)=0

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Protected Member Functions

- **SimulationObject** (const **Reference** &ref)

  Creates a SimulationObject with the specified **Reference**(p. 378).

- **SimulationObject** (const **Referencable** &ref)

  Creates a SimulationObject from the specified **Referencable**(p. 375).

- **SimulationObject** (const **DataObject** &d)

  Creates a SimulationObject from the specified **DataObject**(p. 145).

4.145.1 Detailed Description

An abstract base class for all SimulationObjects.

Author:
Per Alexius

Date:
2006/03/06 09:18:12

4.145.2 Constructor & Destructor Documentation

4.145.2.1 SimulationObject::SimulationObject (const Reference & *ref*) [protected]

Creates a SimulationObject with the specified **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) for the object to be created.

4.145.2.2 SimulationObject::SimulationObject (const Referencable & *ref*) [protected]

Creates a SimulationObject from the specified **Referencable**(p. 375).

Parameters:
ref A **Referencable**(p. 375) for the object to be created.

4.145.2.3 SimulationObject::SimulationObject (const DataObject & *d*) [protected]

Creates a SimulationObject from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.145.3 Member Function Documentation

4.145.3.1 virtual void SimulationObject::extract (Buffer & *b*) const [pure virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implemented in **Activity** (p. 22), **Order** (p. 310), **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 514), **DefendOrder** (p. 166), **AmbushOrder** (p. 43), **AgencyTeam** (p. 37), **CustomAgencyTeam** (p. 137), **City** (p. 93), **Disease** (p. 169), **CityDistribution** (p. 96), **UniformDistribution** (p. 543), **NormalDistribution** (p. 305), **Element** (p. 182), **Faction** (p. 214), **SquarePartitioner** (p. 455), **ModelParameters** (p. 302), **TemplateParameterGroup< ENUM, SIZE >** (p. 511), **Region** (p. 388), **Scenario** (p. 399), **Simulation** (p. 426), **ConstantStepper** (p. 123), **Unit** (p. 556), **TemplateParameterGroup< eInsurgentModelP, eNumInsurgentModelP >** (p. 511), **TemplateParameterGroup< NOTYPE, 0 >** (p. 511), and **TemplateParameterGroup< eFoodModelP, eNumFoodModelP >** (p. 511).

4.145.3.2 virtual void SimulationObject::reset (const DataObject & *d*) [pure virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Implemented in **Activity** (p. 24), **Order** (p. 312), **CustomPVModification** (p. 143), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 167), **AmbushOrder** (p. 44), **AgencyTeam** (p. 38), **CustomAgencyTeam** (p. 138), **Disease** (p. 170), **Distribution** (p. 175), **CityDistribution** (p. 96), **NormalDistribution** (p. 306), **Element** (p. 183), **Faction** (p. 215), **SquarePartitioner** (p. 456), **ModelParameters** (p. 302), **TemplateParameterGroup**< **ENUM, SIZE** > (p. 512), **Region** (p. 388), **Scenario** (p. 399), **Simulation** (p. 427), **ConstantStepper** (p. 123), **Unit** (p. 560), **TemplateParameterGroup**< **eInsurgentModelP, eNumInsurgentModelP** > (p. 512), **TemplateParameterGroup**< **NOTYPE, 0** > (p. 512), and **TemplateParameterGroup**< **eFoodModelP, eNumFoodModelP** > (p. 512).

4.145.3.3 virtual void SimulationObject::update (const Update & *u*) [pure virtual]

Updates this object.

Parameters:

u The **Update**(p. 567) to update this object with.

Implemented in **Disease** (p. 170), **Distribution** (p. 175), **CityDistribution** (p. 97), **NormalDistribution** (p. 306), **SquarePartitioner** (p. 456), **ModelParameters** (p. 303), **Region** (p. 388), **ConstantStepper** (p. 123), and **UpdatableSOAdapter** (p. 566).

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

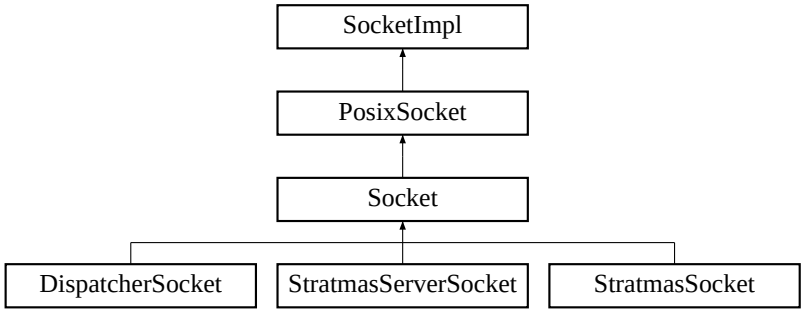
- SimulationObject.h
- SimulationObject.cpp

4.146 Socket Class Reference

C++ wrapper around a C socket.

`#include <Socket.h>`

Inheritance diagram for Socket::



Public Member Functions

- **Socket** ()
  Constructor.
- virtual ~**Socket** ()
  Destructor.

4.146.1 Detailed Description

C++ wrapper around a C socket.

Author:
Per Alexius

Date **Date**
2006/07/03 14:18:23

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

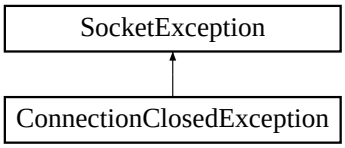
- Socket.h
- Socket.cpp

4.147 SocketException Class Reference

Exception used by **Socket**(p. 432) class.

`#include <SocketException.h>`

Inheritance diagram for SocketException::



Public Member Functions

- **SocketException** (std::string s)
  Creates a SocketException with the specified message.
- ~**SocketException** ()
  Destructor.
- const std::string **description** ()
  Accessor for the message.
- const std::string **what** ()
  Complementary accessor for the message.

Private Attributes

- std::string **mStr**
  The error message.

4.147.1 Detailed Description

Exception used by **Socket**(p. 432) class.

Author:
Per Alexius

Date **Date**
2006/07/21 13:35:29

4.147.2 Constructor & Destructor Documentation

4.147.2.1 SocketException::SocketException (std::string *s*) `[inline]`

Creates a SocketException with the specified message.

Parameters:
s The message.

4.147.3 Member Function Documentation

4.147.3.1 const std::string SocketException::description () [inline]

Accessor for the message.

Returns:
The message.

4.147.3.2 const std::string SocketException::what () [inline]

Complementary accessor for the message.

Returns:
The message.

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

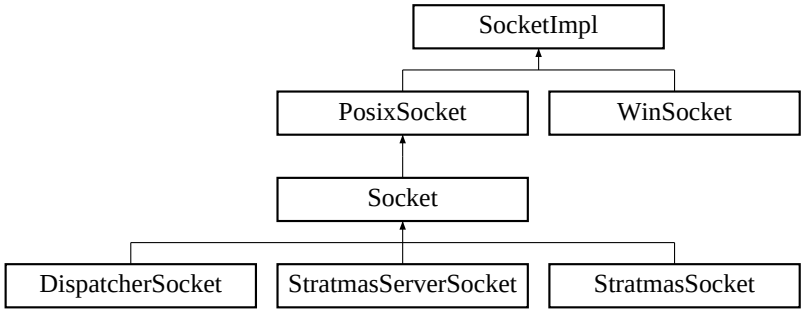
- SocketException.h

4.148 SocketImpl Class Reference

C++ socket implementation interface.

`#include <SocketImpl.h>`

Inheritance diagram for SocketImpl::



Public Member Functions

- virtual bool **create** ()=0
- virtual bool **bind** (const char *host, int port)=0
- virtual bool **listen** () const =0
- virtual bool **accept** (**Socket** &newSock) const =0
- virtual bool **connect** (const std::string host, const int port)=0
- virtual bool **close** ()=0
- virtual bool **send** (const void *msg, unsigned int len) const =0
- virtual int **recv** (void *msg, unsigned int len) const =0
- virtual int **recvf** (void *msg, int len) const =0
- virtual void **set_non_blocking** (const bool b)=0
- virtual bool **valid** () const =0
- virtual std::string **address** () const =0

4.148.1 Detailed Description

C++ socket implementation interface.

Author:
Daniel Ahlin

Date **Date**
2006/07/03 14:18:23

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

- SocketImpl.h

4.149 SOFactory Class Reference

Factory for SimulationObjects.

`#include <SOFactory.h>`

Static Public Member Functions

- **SimulationObject** * **createSimulationObject** (const **DataObject** &d, int64_t initiator=-1)

  Creates a **SimulationObject**(p. 429) *from the provided* **DataObject**(p. 145).

- **SimulationObject** * **createSimulationObject** (const **Reference** &ref, const **Type** &type)

  Creates a **SimulationObject**(p. 429) *of the provided* **Type**(p. 528) *with the provided* **Reference**(p. 378) *by first creating the corresponing* **DataObject**(p. 145) *and then calling* **createSimulationObject()**(p. 446) *with the* **DataObject**(p. 145) *as parameter.*

- void **createOptionalSimpleIn** (**DataObject** &d, const std::string &idToAdd, int64_t initiator=-1)

  Since ValueType descendants have no corresponding **SimulationObject**(p. 429) *we can not create an actual* **SimulationObject**(p. 429) *so we create the corresponding* **DataObject**(p. 145) *and register it.*

- void **createSimpleInList** (const **Reference** &ref, const **Type** &type)

  Since ValueType descendants have no corresponding **SimulationObject**(p. 429) *we can not create an actual* **SimulationObject**(p. 429) *so we create a* **DataObject**(p. 145) *with the provided reference and type and calls* **createSimple()**(p. 445).

- void **createSimple** (**DataObject** &d, int64_t initiator=-1)

  Since ValueType descendants have no corresponding **SimulationObject**(p. 429) *we can not create an actual* **SimulationObject**(p. 429) *so we simply add the provided* **DataObject**(p. 145) *to the* **DataObject**(p. 145) *referenced by its scope and registers it.*

- void **removeSimulationObject** (**SimulationObject** *o, int64_t initiator=-1)

  Removes a **SimulationObject**(p. 429).

- void **addListener** (const **Reference** &ref, **SOFactoryListener** &listener)

  Adds a listener.

- void **removeListener** (const **Reference** &ref, **SOFactoryListener** &listener)

  Removes a listener.

- void **simulationObjectRemoved** (const **Reference** &ref, int64_t initiator)

  This function should be called when the server has removed a ValueType descendant on its own initiaive. Since there is no actual **SimulationObject**(p. 429) *to remove we deregister and remove the corresponding* **DataObject**(p. 145).

- **SimulationObject** * **simulationObjectReplaced** (**DataObject** &newObj, int64_t initiator)

  This function should be called when the server has replaced an object on its own initiaive.

- **SimulationObject** * **createAmbushOrder** (const **DataObject** &d)
  Creates a **AmbushOrder**(p. 41) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createAttackOrder** (const **DataObject** &d)
  Creates a **AttackOrder**(p. 55) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createCity** (const **DataObject** &d)
  Creates a **City**(p. 91) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createCityDistribution** (const **DataObject** &d)
  Creates a **CityDistribution**(p. 95) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createCommonScenario** (const **DataObject** &d)
  Creates a CommonScenario object from the specified **DataObject**(p. 145).
- **SimulationObject** * **createCommonSimulation** (const **DataObject** &d)
  Creates a **CommonSimulation**(p. 109) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createConstantStepper** (const **DataObject** &d)
  Creates a **ConstantStepper**(p. 122) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createCustomAgencyTeam** (const **DataObject** &d)
  Creates a **CustomAgencyTeam**(p. 135) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createCustomPVModification** (const **DataObject** &d)
  Creates a **CustomPVModification**(p. 139) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createDefendOrder** (const **DataObject** &d)
  Creates a **DefendOrder**(p. 165) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createDisease** (const **DataObject** &d)
  Creates a **Disease**(p. 168) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createEthnicFaction** (const **DataObject** &d)
  Creates a **EthnicFaction**(p. 209) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createFoodAgencyTeam** (const **DataObject** &d)
  Creates a FactionRelation object from the specified **DataObject**(p. 145). *Creates a* **FoodAgencyTeam**(p. 219) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createGoToOrder** (const **DataObject** &d)
  Creates a **GoToOrder**(p. 225) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createHealthAgencyTeam** (const **DataObject** &d)
  Creates a **HealthAgencyTeam**(p. 260) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createMilitaryFaction** (const **DataObject** &d)
  Creates a **MilitaryFaction**(p. 300) *object from the specified* **DataObject**(p. 145).
- **SimulationObject** * **createModelParameters** (const **DataObject** &d)

Creates a **ModelParameters**(p. 301) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createNormalDistribution** (const **DataObject** &d)

  Creates a **NormalDistribution**(p. 304) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createParameterGroup** (const **DataObject** &d)

  Creates a **ParameterGroup**(p. 314) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createPoliceAgencyTeam** (const **DataObject** &d)

  Creates a **PoliceAgencyTeam**(p. 324) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createRandomUniformDistribution** (const **DataObject** &d)

  Creates a **RandomUniformDistribution**(p. 373) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createRegion** (const **DataObject** &d)

  Creates a **Region**(p. 386) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createShelterAgencyTeam** (const **DataObject** &d)

  Creates a **ShelterAgencyTeam**(p. 422) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createSquarePartitioner** (const **DataObject** &d)

  Creates a **SquarePartitioner**(p. 454) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createTerroristAttackOrder** (const **DataObject** &d)

  Creates a **TerroristAttackOrder**(p. 513) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createUniformDistribution** (const **DataObject** &d)

  Creates a **UniformDistribution**(p. 542) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createUnit** (const **DataObject** &d)

  Creates a **Unit**(p. 546) *object from the specified* **DataObject**(p. 145).

- **SimulationObject** * **createWaterAgencyTeam** (const **DataObject** &d)

  Creates a **WaterAgencyTeam**(p. 574) *object from the specified* **DataObject**(p. 145).

Static Private Member Functions

- void **fireObjectAdded** (const **Reference** &ref, int64_t initiator)

  Fires an objectAdded event.

- void **fireObjectRemoved** (const **Reference** &ref, int64_t initiator)

  Fires an objectRemoved event.

Static Private Attributes

- std::map< std::string, **SimulationObject** *(*)(const **DataObject** &)> **sCreatorMap**

  Maps the name of a type to the function used to create a **SimulationObject**(p. 429) *of that type from a* **DataObject**(p. 145).

- std::map< const **Reference** *, std::set< **SOFactoryListener** * > > **mListeners**

  The listeners listening to object creation and removal events.

4.149.1 Detailed Description

Factory for SimulationObjects.

Author:
Per Alexius

Date:
2006/10/02 16:01:49

4.149.2 Member Function Documentation

4.149.2.1 void SOFactory::addListener (const Reference & *ref*, SOFactoryListener & *listener*) [inline, static]

Adds a listener.

Parameters:
ref **Reference**(p. 378) to the container to listen to.
listener The listener to add.

4.149.2.2 SimulationObject * SOFactory::createAmbushOrder (const DataObject & *d*) [static]

Creates a **AmbushOrder**(p. 41) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **AmbushOrder**(p. 41) object from.

Returns:
The newly created **AmbushOrder**(p. 41) object.

4.149.2.3 SimulationObject * SOFactory::createAttackOrder (const DataObject & *d*) [static]

Creates a **AttackOrder**(p. 55) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **AttackOrder**(p. 55) object from.

Returns:
The newly created **AttackOrder**(p. 55) object.

4.149.2.4 SimulationObject * SOFactory::createCity (const DataObject & *d*) [static]

Creates a **City**(p. 91) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **City**(p. 91) object from.

Returns:
The newly created **City**(p. 91) object.

4.149.2.5 SimulationObject * SOFactory::createCityDistribution (const DataObject & *d*) [static]

Creates a **CityDistribution**(p. 95) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **CityDistribution**(p. 95) object from.

Returns:
The newly created **CityDistribution**(p. 95) object.

4.149.2.6 SimulationObject * SOFactory::createCommonScenario (const DataObject & *d*) [static]

Creates a CommonScenario object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this CommonScenario object from.

Returns:
The newly created CommonScenario object.

4.149.2.7 SimulationObject * SOFactory::createCommonSimulation (const DataObject & *d*) [static]

Creates a **CommonSimulation**(p. 109) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **CommonSimulation**(p. 109) object from.

Returns:
The newly created **CommonSimulation**(p. 109) object.

4.149.2.8 SimulationObject * SOFactory::createConstantStepper (const DataObject & d) [static]

Creates a **ConstantStepper**(p. 122) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **ConstantStepper**(p. 122) object from.

Returns:
The newly created **ConstantStepper**(p. 122) object.

4.149.2.9 SimulationObject * SOFactory::createCustomAgencyTeam (const DataObject & d) [static]

Creates a **CustomAgencyTeam**(p. 135) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **CustomAgencyTeam**(p. 135) object from.

Returns:
The newly created **CustomAgencyTeam**(p. 135) object.

4.149.2.10 SimulationObject * SOFactory::createCustomPVModification (const DataObject & d) [static]

Creates a **CustomPVModification**(p. 139) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **CustomPVModification**(p. 139) object from.

Returns:
The newly created **CustomPVModification**(p. 139) object.

4.149.2.11 SimulationObject * SOFactory::createDefendOrder (const DataObject & d) [static]

Creates a **DefendOrder**(p. 165) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **DefendOrder**(p. 165) object from.

Returns:
The newly created **DefendOrder**(p. 165) object.

4.149.2.12 SimulationObject * SOFactory::createDisease (const DataObject & *d*) [static]

Creates a **Disease**(p. 168) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **Disease**(p. 168) object from.

Returns:
The newly created **Disease**(p. 168) object.

4.149.2.13 SimulationObject * SOFactory::createEthnicFaction (const DataObject & *d*) [static]

Creates a **EthnicFaction**(p. 209) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **EthnicFaction**(p. 209) object from.

Returns:
The newly created **EthnicFaction**(p. 209) object.

4.149.2.14 SimulationObject * SOFactory::createFoodAgencyTeam (const DataObject & *d*) [static]

Creates a FactionRelation object from the specified **DataObject**(p. 145). Creates a **FoodAgencyTeam**(p. 219) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **FoodAgencyTeam**(p. 219) object from.

Returns:
The newly created **FoodAgencyTeam**(p. 219) object.

4.149.2.15 SimulationObject * SOFactory::createGoToOrder (const DataObject & *d*) [static]

Creates a **GoToOrder**(p. 225) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **GoToOrder**(p. 225) object from.

Returns:
The newly created **GoToOrder**(p. 225) object.

4.149.2.16 SimulationObject * SOFactory::createHealthAgencyTeam (const DataObject & d) [static]

Creates a **HealthAgencyTeam**(p. 260) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **HealthAgencyTeam**(p. 260) object from.

Returns:
The newly created **HealthAgencyTeam**(p. 260) object.

4.149.2.17 SimulationObject * SOFactory::createMilitaryFaction (const DataObject & d) [static]

Creates a **MilitaryFaction**(p. 300) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **MilitaryFaction**(p. 300) object from.

Returns:
The newly created **MilitaryFaction**(p. 300) object.

4.149.2.18 SimulationObject * SOFactory::createModelParameters (const DataObject & d) [static]

Creates a **ModelParameters**(p. 301) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **ModelParameters**(p. 301) object from.

Returns:
The newly created **ModelParameters**(p. 301) object.

4.149.2.19 SimulationObject * SOFactory::createNormalDistribution (const DataObject & d) [static]

Creates a **NormalDistribution**(p. 304) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **NormalDistribution**(p. 304) object from.

Returns:
The newly created **NormalDistribution**(p. 304) object.

4.149.2.20 void SOFactory::createOptionalSimpleIn (DataObject & *d*, const std::string & *idToAdd*, int64_t *initiator* = -1) [static]

Since ValueType descendants have no corresponding **SimulationObject**(p. 429) we can not create an actual **SimulationObject**(p. 429) so we create the corresponding **DataObject**(p. 145) and register it.

This function should only be called when the server creates ValueType descendants that are optional.

Parameters:

d The **DataObject**(p. 145) to create a **SimulationObject**(p. 429) in.

idToAdd The identifier of the object to add.

initiator The id of the initiator of the creation.

4.149.2.21 SimulationObject * SOFactory::createParameterGroup (const DataObject & *d*) [static]

Creates a **ParameterGroup**(p. 314) object from the specified **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this **ParameterGroup**(p. 314) object from.

Returns:

The newly created **ParameterGroup**(p. 314) object.

4.149.2.22 SimulationObject * SOFactory::createPoliceAgencyTeam (const DataObject & *d*) [static]

Creates a **PoliceAgencyTeam**(p. 324) object from the specified **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this **PoliceAgencyTeam**(p. 324) object from.

Returns:

The newly created **PoliceAgencyTeam**(p. 324) object.

4.149.2.23 SimulationObject * SOFactory::createRandomUniformDistribution (const DataObject & *d*) [static]

Creates a **RandomUniformDistribution**(p. 373) object from the specified **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this **RandomUniformDistribution**(p. 373) object from.

Returns:

The newly created **RandomUniformDistribution**(p. 373) object.

4.149.2.24 SimulationObject * SOFactory::createRegion (const DataObject & *d*) [static]

Creates a **Region**(p. 386) object from the specified **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this **Region**(p. 386) object from.

Returns:

The newly created **Region**(p. 386) object.

4.149.2.25 SimulationObject * SOFactory::createShelterAgencyTeam (const DataObject & *d*) [static]

Creates a **ShelterAgencyTeam**(p. 422) object from the specified **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this **ShelterAgencyTeam**(p. 422) object from.

Returns:

The newly created **ShelterAgencyTeam**(p. 422) object.

4.149.2.26 void SOFactory::createSimple (DataObject & *d*, int64_t *initiator* = -1) [static]

Since ValueType descendants have no corresponding **SimulationObject**(p. 429) we can not create an actual **SimulationObject**(p. 429) so we simply add the provided **DataObject**(p. 145) to the **DataObject**(p. 145) referenced by its scope and registers it.

This function should only be called when the server creates ValueType descendants on its own initiative.

Parameters:

d The **DataObject**(p. 145) representing the ValueType desendant.

initiator The id of the initiator of the creation.

4.149.2.27 void SOFactory::createSimpleInList (const Reference & *ref*, const Type & *type*) [static]

Since ValueType descendants have no corresponding **SimulationObject**(p. 429) we can not create an actual **SimulationObject**(p. 429) so we create a **DataObject**(p. 145) with the provided reference and type and calls **createSimple()**(p. 445).

Parameters:

ref The **Reference**(p. 378) to the object to be created.

type The type of the object to be created.

4.149.2.28 SimulationObject * SOFactory::createSimulationObject (const Reference & *ref*, const Type & *type*) [static]

Creates a **SimulationObject**(p. 429) of the provided **Type**(p. 528) with the provided **Reference**(p. 378) by first creating the corresponing **DataObject**(p. 145) and then calling **createSimulationObject()**(p. 446) with the **DataObject**(p. 145) as parameter.

This function should only be called when the server creates optional objects or objects in lists on its own initaitve.

Parameters:

ref The **Reference**(p. 378) to create a **SimulationObject**(p. 429) for.

type The **Type**(p. 528) of **SimulationObject**(p. 429) to create.

Returns:

The newly created **SimulationObject**(p. 429).

4.149.2.29 SimulationObject * SOFactory::createSimulationObject (const DataObject & *d*, int64_t *initiator* = -1) [static]

Creates a **SimulationObject**(p. 429) from the provided **DataObject**(p. 145).

This function should only be called for DataObjects which type is a non ValueType descendant.

Parameters:

d The **DataObject**(p. 145) to create a **SimulationObject**(p. 429) from.

initiator The id of the initiator of the creation.

Returns:

The newly created **SimulationObject**(p. 429).

4.149.2.30 SimulationObject * SOFactory::createSquarePartitioner (const DataObject & *d*) [static]

Creates a **SquarePartitioner**(p. 454) object from the specified **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this **SquarePartitioner**(p. 454) object from.

Returns:

The newly created **SquarePartitioner**(p. 454) object.

4.149.2.31 SimulationObject * SOFactory::createTerroristAttackOrder (const DataObject & *d*) [static]

Creates a **TerroristAttackOrder**(p. 513) object from the specified **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to create this **TerroristAttackOrder**(p. 513) object from.

Returns:

The newly created **TerroristAttackOrder**(p. 513) object.

4.149.2.32 SimulationObject * SOFactory::createUniformDistribution (const DataObject & *d*) [static]

Creates a **UniformDistribution**(p. 542) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **UniformDistribution**(p. 542) object from.

Returns:
The newly created **UniformDistribution**(p. 542) object.

4.149.2.33 SimulationObject * SOFactory::createUnit (const DataObject & *d*) [static]

Creates a **Unit**(p. 546) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **Unit**(p. 546) object from.

Returns:
The newly created **Unit**(p. 546) object.

4.149.2.34 SimulationObject * SOFactory::createWaterAgencyTeam (const DataObject & *d*) [static]

Creates a **WaterAgencyTeam**(p. 574) object from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this **WaterAgencyTeam**(p. 574) object from.

Returns:
The newly created **WaterAgencyTeam**(p. 574) object.

4.149.2.35 void SOFactory::fireObjectAdded (const Reference & *ref*, int64_t *initiator*) [static, private]

Fires an objectAdded event.

Both the **SimulationObject**(p. 429) and the corresponding **DataObject**(p. 145) exists and are registered when this call occurs.

Parameters:
ref The **Reference**(p. 378) to the added object.

initiator The id of the initiator of the event.

4.149.2.36 void SOFactory::fireObjectRemoved (const Reference & *ref*, int64_t *initiator*) [static, private]

Fires an objectRemoved event.

When this function is called the **SimulationObject**(p. 429) is already deleted and deregistered. The corresponding **DataObject**(p. 145) does still exist and is still registered.

Parameters:

ref The **Reference**(p. 378) to the removed object.

initiator The id of the initiator of the event.

4.149.2.37 void SOFactory::removeListener (const Reference & *ref*, SOFactoryListener & *listener*) [static]

Removes a listener.

Parameters:

ref **Reference**(p. 378) to the container that has been listened to.

listener The listener to remove.

4.149.2.38 void SOFactory::removeSimulationObject (SimulationObject * *o*, int64_t *initiator* = -1) [static]

Removes a **SimulationObject**(p. 429).

Parameters:

o The **SimulationObject**(p. 429) to remove.

initiator The id of the initiator of the removal.

4.149.2.39 void SOFactory::simulationObjectRemoved (const Reference & *ref*, int64_t *initiator*) [static]

This function should be called when the server has removed a ValueType descendant on its own initiaive. Since there is no actual **SimulationObject**(p. 429) to remove we deregister and remove the corresponding **DataObject**(p. 145).

Parameters:

ref The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the removal.

4.149.2.40 SimulationObject * SOFactory::simulationObjectReplaced (DataObject & *newObj*, int64_t *initiator*) [static]

This function should be called when the server has replaced an object on its own initiaive.

If the removed object is a ValueType descendant no new **SimulationObject**(p. 429) is created. Otherwise the **SimulationObject**(p. 429) created from the replacing **DataObject**(p. 145) is returned.

Parameters:

newObj The **DataObject**(p. 145) to replace the old object with.

initiator The id of the initiator of the removal.

Returns:

The new **SimulationObject**(p. 429) or null if newObject is a ValueType descendant.

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

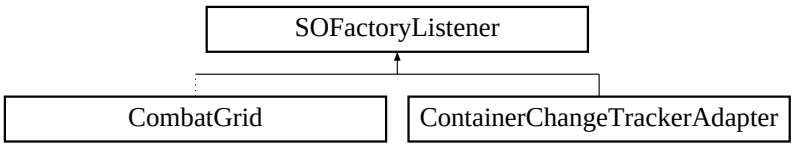
- SOFactory.h
- SOFactory.cpp

4.150 SOFactoryListener Class Reference

SOFactoryListener is a pure virtual class defining the interface for objects that listen to the **SOFactory**(p. 436).

`#include <SOFactoryListener.h>`

Inheritance diagram for SOFactoryListener::



Public Member Functions

- virtual ~**SOFactoryListener** ()
  Destructor.
- virtual void **objectAdded** (const **Reference** &ref, int64_t initiator)=0
  Called when an object has been added by the **SOFactory**(p. 436).
- virtual void **objectRemoved** (const **Reference** &ref, int64_t initiator)=0
  Called when an object has been removed by the **SOFactory**(p. 436).

4.150.1 Detailed Description

SOFactoryListener is a pure virtual class defining the interface for objects that listen to the **SOFactory**(p. 436).

Author:
Per Alexius

Date:
2006/07/03 14:18:23

4.150.2 Member Function Documentation

4.150.2.1 virtual void SOFactoryListener::objectAdded (const Reference & *ref*, int64_t *initiator*) [pure virtual]

Called when an object has been added by the **SOFactory**(p. 436).

Both the **SimulationObject**(p. 429) and the corresponding **DataObject**(p. 145) exists and are registered when this call occurs.

Parameters:
ref The **Reference**(p. 378) to the object that was added.
initiator The id of the initiator of the event.

Implemented in **ContainerChangeTrackerAdapter** (p. 127), and **CombatGrid** (p. 106).

4.150.2.2 virtual void SOFactoryListener::objectRemoved (const Reference & *ref*, int64_t *initiator*) [pure virtual]

Called when an object has been removed by the **SOFactory**(p. 436).

When this function is called the **SimulationObject**(p. 429) is already deleted and deregistered. The corresponding **DataObject**(p. 145) does still exist and is still registered.

Parameters:

ref The **Reference**(p. 378) to the object that is removed

initiator The id of the initiator of the event.

Implemented in **ContainerChangeTrackerAdapter** (p. 128), and **CombatGrid** (p. 107).

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

- SOFactoryListener.h

4.151 SOMapper Class Reference

This class is used to map References to their corresponding **SimulationObject**(p. 429).

`#include <SOMapper.h>`

Static Public Member Functions

- void **reg** (**SimulationObject** *c)

  Registers the provided **SimulationObject**(p. 429) *with this SOMapper.*

- void **dereg** (const **Reference** &ref)

  Deregisters the provided **Reference**(p. 378) *from this SOMapper.*

- **SimulationObject** * **map** (const **Reference** &ref)

  Maps the provided **Reference**(p. 378) *to its corresponding* **SimulationObject**(p. 429).

- void **clear** ()

  Erases all mappings.

- void **extract** (**Buffer** &b)

  Extracts data from all mapped objects to the provided **Buffer**(p. 67).

Static Private Attributes

- SOMap **mMap**

  Contains the mappings between **Reference**(p. 378) *and* **SimulationObject**(p. 429).

Friends

- std::ostream & **operator**<< (std::ostream &o, const **SOMapper** &m)

  For debugging purposes.

4.151.1 Detailed Description

This class is used to map References to their corresponding **SimulationObject**(p. 429).

Author:
Per Alexius

Date **Date**
2006/07/03 14:18:23

4.151.2 Member Function Documentation

4.151.2.1 void SOMapper::dereg (const Reference & *ref*) [inline, static]

Deregisters the provided **Reference**(p. 378) from this SOMapper.

Parameters:
ref The **Reference**(p. 378) to deregister.

4.151.2.2 void SOMapper::extract (Buffer & *b*) [static]

Extracts data from all mapped objects to the provided **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

4.151.2.3 SimulationObject* SOMapper::map (const Reference & *ref*) [inline, static]

Maps the provided **Reference**(p. 378) to its corresponding **SimulationObject**(p. 429).

Parameters:
ref The **Reference**(p. 378) to find a **SimulationObject**(p. 429) for.

Returns:
The **SimulationObject**(p. 429) for the provided **Reference**(p. 378) or null if no such **SimulationObject**(p. 429) was found..

4.151.2.4 void SOMapper::reg (SimulationObject * *c*) [static]

Registers the provided **SimulationObject**(p. 429) with this SOMapper.

Parameters:
c The **SimulationObject**(p. 429) to register.

4.151.3 Friends And Related Function Documentation

4.151.3.1 std::ostream& operator<< (std::ostream & *o*, const SOMapper & *m*) [friend]

For debugging purposes.

Parameters:
o The stream to write to.
m The SOMapper to print.

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

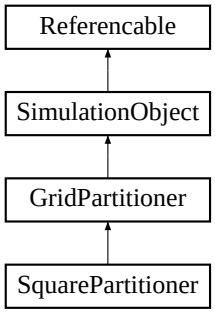
- SOMapper.h
- SOMapper.cpp

4.152 SquarePartitioner Class Reference

A **GridPartitioner**(p. 254) that creates a grid with square cells.

`#include <GridPartitioner.h>`

Inheritance diagram for SquarePartitioner::



Public Member Functions

- **SquarePartitioner** (const **DataObject** &d)

  Creates a SquarePartitioner from the provided **DataObject**(p. 145).

- virtual ~**SquarePartitioner** ()

  Destructor.

- void **update** (const **Update** &u)

  Square partitioners can not be updated so calling this function is an erroneous behavior.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- **Grid** * **createGrid** (const **Map** &m, int numEthnicFactions) const

  Creates a **Grid**(p. 227).

- double **cellSideMeters** ()

  Get the length of the cell side in meters.

Private Attributes

- double **mCellSideMeters**

  Length of the timestep.

4.152.1 Detailed Description

A **GridPartitioner**(p. 254) that creates a grid with square cells.

Author:
Per Alexius

Date:
2006/03/06 14:23:07

4.152.2 Constructor & Destructor Documentation

4.152.2.1 SquarePartitioner::SquarePartitioner (const DataObject & *d*)

Creates a SquarePartitioner from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.152.3 Member Function Documentation

4.152.3.1 double SquarePartitioner::cellSideMeters () [inline]

Get the length of the cell side in meters.

Returns:
The length of the cell side in meters.

4.152.3.2 Grid * SquarePartitioner::createGrid (const Map & *m*, int *numEthnicFactions*) const [virtual]

Creates a **Grid**(p. 227).

Parameters:
m The **Map**(p. 292) to create the **Grid**(p. 227) for.
numEthnicFactions The number of ethnic factions.

Returns:
The newly created **Grid**(p. 227).

Implements **GridPartitioner** (p. 255).

4.152.3.3 void SquarePartitioner::extract (Buffer & *b*) const [inline, virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.152.3.4 void SquarePartitioner::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

4.152.3.5 void SquarePartitioner::update (const Update & *u*) [virtual]

Square partitioners can not be updated so calling this function is an erroneous behavior.

Parameters:

u The **Update**(p. 567) to update this object with.

Implements **SimulationObject** (p. 431).

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

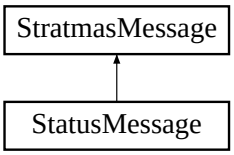
- GridPartitioner.h
- GridPartitioner.cpp

4.153 StatusMessage Class Reference

Class representing the StatusMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for StatusMessage::



Public Member Functions

- **StatusMessage** (const std::string &type)
  Constructs a StatusMessage with the specified type.
- void **addError** (const **Error** &e)
  Adds an **Error**(p. 205) *to this message.*
- void **toXML** (std::ostream &o) const
  Produces the XML representation of this message.

Private Attributes

- const std::string **mType**
  String holding the type of message this message is a response to.
- std::vector< **Error** > **mErrors**
  Vector containing all errors that should be reported in this status message.

4.153.1 Detailed Description

Class representing the StatusMessage.

Author:
Per Alexius

Date **Date**
2006/03/06 14:23:12

4.153.2 Constructor & Destructor Documentation

4.153.2.1 StatusMessage::StatusMessage (const std::string & *type*) [inline]

Constructs a StatusMessage with the specified type.

Parameters:
type The type of the StatusMessage to create.

4.153.3 Member Function Documentation

4.153.3.1 void StatusMessage::addError (const Error & *e*) [inline]

Adds an **Error**(p. 205) to this message.

Parameters:
e The **Error**(p. 205) to be added

4.153.3.2 void StatusMessage::toXML (std::ostream & *o*) const [virtual]

Produces the XML representation of this message.

Parameters:
o The stream to which the message is written

Implements **StratmasMessage** (p. 473).

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

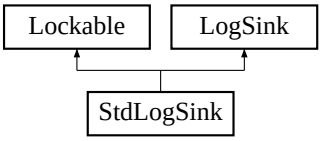
- StratmasMessage.h
- StratmasMessage.cpp

4.154 StdLogSink Class Reference

This class implements **LogSink**(p. 289) using std::cerr for output.

`#include <LogStream.h>`

Inheritance diagram for StdLogSink::



Public Member Functions

- virtual void **sink** (const **LogMessage** *const message)
  Posts the provided message to the log stream.

Private Member Functions

- virtual const std::string **getTimeStamp** () const
  Returns a string representation of the current time.

4.154.1 Detailed Description

This class implements **LogSink**(p. 289) using std::cerr for output.

Author:
Daniel Ahlin

Date **Date**
2006/07/25 14:52:00

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

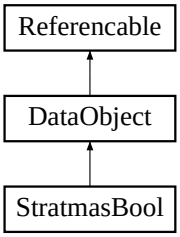
- LogStream.h
- LogStream.cpp

4.155 StratmasBool Class Reference

StratmasBool corresponds to the Boolean type in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for StratmasBool::



Public Member Functions

- **StratmasBool** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **StratmasBool** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- bool **getBool** () const

  Accessor.

- void **setBool** (bool v)

  Mutator.

- **DataObject** & **operator=** (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **StratmasBool** (const **StratmasBool** &c)

  Copy constructor.

Private Attributes

- bool **mValue**

  The value.

4.155.1 Detailed Description

StratmasBool corresponds to the Boolean type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/03/27 09:43:40

4.155.2 Constructor & Destructor Documentation

4.155.2.1 StratmasBool::StratmasBool (const StratmasBool & *c*) [inline, protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.155.2.2 StratmasBool::StratmasBool (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.155.2.3 StratmasBool::StratmasBool (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.155.3 Member Function Documentation

4.155.3.1 ostream & StratmasBool::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.155.3.2 DataObject* StratmasBool::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.155.3.3 bool StratmasBool::getBool () const [inline, virtual]

Accessor.

Returns:

The current value.

Reimplemented from **DataObject** (p. 149).

4.155.3.4 DataObject & StratmasBool::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.155.3.5 void StratmasBool::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.155.3.6 void StratmasBool::setBool (bool *v*) [inline, virtual]

Mutator.

Parameters:

v The new value.

Reimplemented from **DataObject** (p. 152).

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

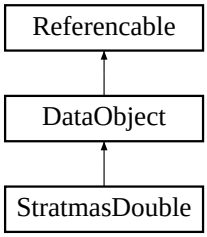
- DataObjectImpl.h
- DataObjectImpl.cpp

4.156 StratmasDouble Class Reference

StratmasDouble corresponds to the Double type in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for StratmasDouble::



Public Member Functions

- **StratmasDouble** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **StratmasDouble** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- double **getDouble** () const

  Accessor.

- void **setDouble** (double v)

  Mutator.

- **DataObject** & **operator**= (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **StratmasDouble** (const **StratmasDouble** &c)

  Copy constructor.

Private Attributes

- double **mValue**

  The value.

4.156.1 Detailed Description

StratmasDouble corresponds to the Double type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/03/27 09:43:40

4.156.2 Constructor & Destructor Documentation

4.156.2.1 StratmasDouble::StratmasDouble (const StratmasDouble & *c*) [inline, protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.156.2.2 StratmasDouble::StratmasDouble (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.156.2.3 StratmasDouble::StratmasDouble (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.156.3 Member Function Documentation

4.156.3.1 ostream & StratmasDouble::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.156.3.2 DataObject* StratmasDouble::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.156.3.3 double StratmasDouble::getDouble () const [inline, virtual]

Accessor.

Returns:

The current value.

Reimplemented from **DataObject** (p. 150).

4.156.3.4 DataObject & StratmasDouble::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.156.3.5 void StratmasDouble::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.156.3.6 void StratmasDouble::setDouble (double *v*) [inline, virtual]

Mutator.

Parameters:

v The new value.

Reimplemented from **DataObject** (p. 152).

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

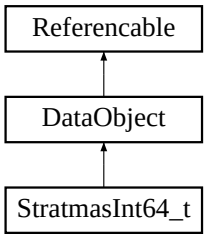
- DataObjectImpl.h
- DataObjectImpl.cpp

4.157 StratmasInt64_t Class Reference

StratmasInt64_t corresponds to the NonNegativeInteger type in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for StratmasInt64_t::



Public Member Functions

- **StratmasInt64_t** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **StratmasInt64_t** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- int64_t **getInt64_t** () const

  Accessor.

- void **setInt64_t** (int64_t v)

  Mutator.

- **DataObject** & **operator=** (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **StratmasInt64_t** (const **StratmasInt64_t** &c)

  Copy constructor.

Private Attributes

- int64_t **mValue**

  The value.

4.157.1 Detailed Description

StratmasInt64_t corresponds to the NonNegativeInteger type in the Stratmas xml schema.

Author:
Per Alexius

Date **Date**
2006/03/27 09:43:40

4.157.2 Constructor & Destructor Documentation

4.157.2.1 StratmasInt64_t::StratmasInt64_t (const StratmasInt64_t & *c*) [inline, protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.157.2.2 StratmasInt64_t::StratmasInt64_t (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.157.2.3 StratmasInt64_t::StratmasInt64_t (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.157.3 Member Function Documentation

4.157.3.1 ostream & StratmasInt64_t::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.157.3.2 DataObject* StratmasInt64_t::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.157.3.3 int64_t StratmasInt64_t::getInt64_t () const [inline, virtual]

Accessor.

Returns:

The current value.

Reimplemented from **DataObject** (p. 150).

4.157.3.4 DataObject & StratmasInt64_t::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.157.3.5 void StratmasInt64_t::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.157.3.6 void StratmasInt64_t::setInt64_t (int64_t *v*) [inline, virtual]

Mutator.

Parameters:

v The new value.

Reimplemented from **DataObject** (p. 152).

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

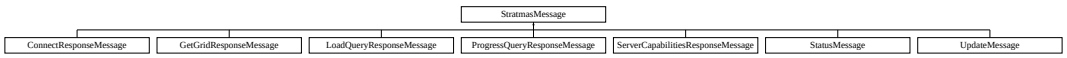
- DataObjectImpl.h
- DataObjectImpl.cpp

4.158 StratmasMessage Class Reference

Abstract base class for all types of StratmasMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for StratmasMessage::



Public Member Functions

- virtual ~**StratmasMessage** ()
  Destructor.
- virtual void **toXML** (std::ostream &o) const =0
  Produces the XML representation of a message. Must be overridden by all subclasses.

Protected Member Functions

- virtual void **openMessage** (std::ostream &o, const std::string &type) const
  Helper for creating the header of the XML representation of a StratmasMessage.
- virtual void **closeMessage** (std::ostream &o) const
  Helper for creating the end of the XML representation of a StratmasMessage.

4.158.1 Detailed Description

Abstract base class for all types of StratmasMessage.

This class also contains some helpers for creating XML representations of StratmasMessages.

Author:
Per Alexius

Date:
2006/03/06 14:23:12

4.158.2 Member Function Documentation

4.158.2.1 void StratmasMessage::closeMessage (std::ostream & *o*) const [protected, virtual]

Helper for creating the end of the XML representation of a StratmasMessage.

Parameters:
o The stream to write to.

4.158.2.2 void StratmasMessage::openMessage (std::ostream & *o*, const std::string & *type*) const [protected, virtual]

Helper for creating the header of the XML representation of a StratmasMessage.

Parameters:

o The stream to write to.

type The type of the message to write a header for.

4.158.2.3 virtual void StratmasMessage::toXML (std::ostream & *o*) const [pure virtual]

Produces the XML representation of a message. Must be overridden by all subclasses.

Parameters:

o The stream to which the message is written

Implemented in **ConnectResponseMessage** (p. 121), **ServerCapabilitiesResponseMessage** (p. 407), **GetGridResponseMessage** (p. 224), **StatusMessage** (p. 458), **UpdateMessage** (p. 571), **ProgressQueryResponseMessage** (p. 346), and **LoadQueryResponseMessage** (p. 284).

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

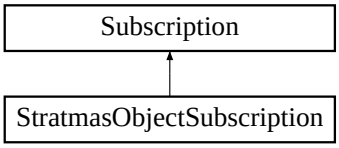
- StratmasMessage.h
- StratmasMessage.cpp

4.159 StratmasObjectSubscription Class Reference

Subscription(p. 500) for individual SimulationObjects.

`#include <Subscription.h>`

Inheritance diagram for StratmasObjectSubscription::



Public Member Functions

- **StratmasObjectSubscription** (DOMElement *n, **Buffer** &buf, int64_t id)
  Creates a subscription from a DOMElement, e.g. an xml representation.
- ~**StratmasObjectSubscription** ()
  Destructor.
- void **getSubscribedData** (std::ostream &o)
  Writes an XML representation of the subscribed data to the provided stream.

Private Attributes

- **ChangeTrackerAdapter** * **mData**
  The **ChangeTrackerAdapter**(p. 82) *for the subscribed object.*

4.159.1 Detailed Description

Subscription(p. 500) for individual SimulationObjects.

Currently only used by the evolver.

Author:
Per Alexius

Date:
2006/07/05 14:49:47

4.159.2 Constructor & Destructor Documentation

4.159.2.1 StratmasObjectSubscription::StratmasObjectSubscription (DOMElement * *n*, Buffer & *buf*, int64_t *id*)

Creates a subscription from a DOMElement, e.g. an xml representation.

Parameters:

n The DOMElement from which this subscription should be created.

buf The **Buffer**(p. 67) from which data should be fetched.

id The id of the **Session**(p. 408) holding this **Subscription**(p. 500).

4.159.3 Member Function Documentation

4.159.3.1 void StratmasObjectSubscription::getSubscribedData (std::ostream & *o*) [virtual]

Writes an XML representation of the subscribed data to the provided stream.

Parameters:

o The stream to write to.

Implements **Subscription** (p. 501).

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

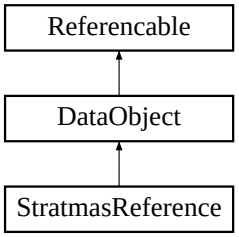
- Subscription.h
- Subscription.cpp

4.160 StratmasReference Class Reference

StratmasReference corresponds to the **Reference**(p. 378) type in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for StratmasReference::



Public Member Functions

- **StratmasReference** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **StratmasReference** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- const **Reference** & **getReference** () const

  Accessor.

- void **setReference** (const **Reference** &v)

  Mutator.

- **DataObject** & **operator=** (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **StratmasReference** (const **StratmasReference** &c)

  Copy constructor.

Private Attributes

- const **Reference** * **mValue**

  The value.

4.160.1 Detailed Description

StratmasReference corresponds to the **Reference**(p. 378) type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/03/27 09:43:40

4.160.2 Constructor & Destructor Documentation

4.160.2.1 StratmasReference::StratmasReference (const StratmasReference & *c*) [inline, protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.160.2.2 StratmasReference::StratmasReference (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.160.2.3 StratmasReference::StratmasReference (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.160.3 Member Function Documentation

4.160.3.1 ostream & StratmasReference::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.160.3.2 DataObject* StratmasReference::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.160.3.3 const Reference& StratmasReference::getReference () const [inline, virtual]

Accessor.

Returns:

The current value.

Reimplemented from **DataObject** (p. 150).

4.160.3.4 DataObject & StratmasReference::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.160.3.5 void StratmasReference::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.160.3.6 void StratmasReference::setReference (const Reference & *v*) [inline, virtual]

Mutator.

Parameters:

v The new value.

Reimplemented from **DataObject** (p. 153).

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

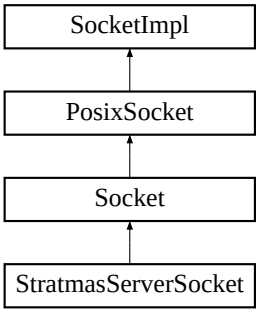
- DataObjectImpl.h
- DataObjectImpl.cpp

4.161 StratmasServerSocket Class Reference

ServerSocket user for listening to incoming stratmas messages.

`#include <StratmasServerSocket.h>`

Inheritance diagram for StratmasServerSocket::



Public Member Functions

- **StratmasServerSocket** (const char *host, int port)

  Creates a socket that will listen to connections on the specified port and hostname.

4.161.1 Detailed Description

ServerSocket user for listening to incoming stratmas messages.

Author:
Per Alexius

Date:
2005/06/13 13:41:15

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

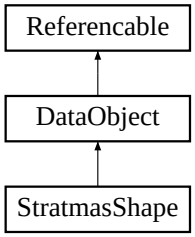
- StratmasServerSocket.h
- StratmasServerSocket.cpp

4.162 StratmasShape Class Reference

StratmasShape corresponds to the **Shape**(p. 412) type in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for StratmasShape::



Public Member Functions

- **StratmasShape** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **StratmasShape** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- **Shape** * **getShape** () const

  Gets a clone of the **Shape**(p. 412). *The caller is responsible for freeing up the memory used by the returned* **Shape**(p. 412).

- **Shape** & **getShapeRef** () const

  Accessor for the actual **Shape**(p. 412) *held by this object.*

- void **setShape** (const **Shape** *v)

  Sets the **Shape**(p. 412) *to a clone of the provided* **Shape**(p. 412).

- **DataObject** & **operator**= (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **StratmasShape** (const **StratmasShape** &c)
  Copy constructor.

Private Attributes

- **Shape** * **mValue**
  The value.

4.162.1 Detailed Description

StratmasShape corresponds to the **Shape**(p. 412) type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/03/27 09:43:40

4.162.2 Constructor & Destructor Documentation

4.162.2.1 StratmasShape::StratmasShape (const StratmasShape & *c*) [protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.162.2.2 StratmasShape::StratmasShape (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.
n The DOMElement to create this **DataObject**(p. 145) from.

4.162.2.3 StratmasShape::StratmasShape (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.
type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.162.3 Member Function Documentation

4.162.3.1 ostream & StratmasShape::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.162.3.2 DataObject* StratmasShape::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.162.3.3 Shape * StratmasShape::getShape () const [virtual]

Gets a clone of the **Shape**(p. 412). The caller is responsible for freeing up the memory used by the returned **Shape**(p. 412).

/return A copy of the **Shape**(p. 412).

Reimplemented from **DataObject** (p. 150).

4.162.3.4 Shape& StratmasShape::getShapeRef () const [inline]

Accessor for the actual **Shape**(p. 412) held by this object.

Returns:

The actual **Shape**(p. 412) held by this object.

4.162.3.5 DataObject & StratmasShape::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.162.3.6 void StratmasShape::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.162.3.7 void StratmasShape::setShape (const Shape * *v*) [virtual]

Sets the **Shape**(p. 412) to a clone of the provided **Shape**(p. 412).

If the provided **Shape**(p. 412) is of different type that the existing - a replaced event is triggered.

/return A copy of the **Shape**(p. 412).

Reimplemented from **DataObject** (p. 153).

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

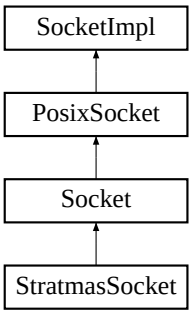
- DataObjectImpl.h
- DataObjectImpl.cpp

4.163 StratmasSocket Class Reference

Socket(p. 432) user for sending and receiving StratmasMessages.

`#include <StratmasSocket.h>`

Inheritance diagram for StratmasSocket::



Public Member Functions

- **StratmasSocket** ()
  Default constructor.
- **StratmasSocket** (std::string host, int port)
  Creates a socket that connects to the specified host and port.
- int64_t **id** () const
  Accessor for the id.
- void **id** (int64_t id)
  Mutator for the id.
- bool **sendStratmasMessage** (const std::string msg) const
  Sends a stratmas message.
- int64_t **recvStratmasHeader** ()
  Receives a StratmasHeader.
- int **recvStratmasMessage** (std::string &outMsg)
  Receives a **StratmasMessage**(p. 472).

Private Attributes

- int64_t **mId**
  The id of this socket.
- int64_t **mLength**
  Contains the length of a message read from the StratmasHeader.

- bool **mReceivedHeader**

  True if we have received a StratmasHeader but not the message the header refers to.

4.163.1 Detailed Description

Socket(p. 432) user for sending and receiving StratmasMessages.

Author:
Per Alexius

Date:
2006/07/03 14:18:23

4.163.2 Constructor & Destructor Documentation

4.163.2.1 StratmasSocket::StratmasSocket (std::string *host*, int *port*)

Creates a socket that connects to the specified host and port.

Parameters:
host The host to connect to.

port The port to connect to.

4.163.3 Member Function Documentation

4.163.3.1 void StratmasSocket::id (int64_t *id*) [inline]

Mutator for the id.

Parameters:
id The new id of this **Socket**(p. 432).

4.163.3.2 int64_t StratmasSocket::id () const [inline]

Accessor for the id.

Returns:
The id of this **Socket**(p. 432).

4.163.3.3 int64_t StratmasSocket::recvStratmasHeader ()

Receives a StratmasHeader.

Returns:
The id contained in the StratmasHeader.

4.163.3.4 int StratmasSocket::recvStratmasMessage (std::string & *outMsg*)

Receives a **StratmasMessage**(p. 472).

Returns:
The number of bytes in the received message.

4.163.3.5 bool StratmasSocket::sendStratmasMessage (const std::string *msg*) const

Sends a stratmas message.

Parameters:
msg The message to send.

Returns:
True if all is ok.

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

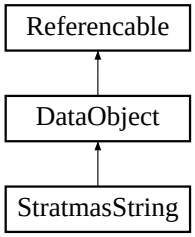
- StratmasSocket.h
- StratmasSocket.cpp

4.164 StratmasString Class Reference

StratmasString corresponds to the String type in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for StratmasString::



Public Member Functions

- **StratmasString** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **StratmasString** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- std::string **getString** () const

  Accessor.

- void **setString** (const std::string &v)

  Mutator.

- **DataObject** & **operator=** (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **StratmasString** (const **StratmasString** &c)

  Copy constructor.

Private Attributes

- std::string **mValue**

  The value.

4.164.1 Detailed Description

StratmasString corresponds to the String type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/03/27 09:43:40

4.164.2 Constructor & Destructor Documentation

4.164.2.1 StratmasString::StratmasString (const StratmasString & *c*) [inline, protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.164.2.2 StratmasString::StratmasString (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.164.2.3 StratmasString::StratmasString (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.164.3 Member Function Documentation

4.164.3.1 ostream & StratmasString::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.164.3.2 DataObject* StratmasString::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.164.3.3 std::string StratmasString::getString () const [inline, virtual]

Accessor.

Returns:

The current value.

Reimplemented from **DataObject** (p. 150).

4.164.3.4 DataObject & StratmasString::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.164.3.5 void StratmasString::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.164.3.6 void StratmasString::setString (const std::string & *v*) [inline, virtual]

Mutator.

Parameters:

v The new value.

Reimplemented from **DataObject** (p. 153).

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

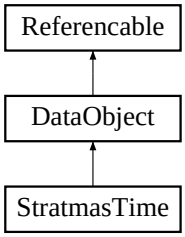
- DataObjectImpl.h
- DataObjectImpl.cpp

4.165 StratmasTime Class Reference

StratmasTime corresponds to the Timestamp and Duration types in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for StratmasTime::



Public Member Functions

- **StratmasTime** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **StratmasTime** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- **Time getTime** () const

  Accessor.

- void **setTime** (**Time** v)

  Mutator.

- **DataObject** & **operator=** (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **StratmasTime** (const **StratmasTime** &c)

  Copy constructor.

Private Attributes

- **Time mValue**

  The value.

4.165.1 Detailed Description

StratmasTime corresponds to the Timestamp and Duration types in the Stratmas xml schema.

Author:
Per Alexius

Date Date
2006/03/27 09:43:40

4.165.2 Constructor & Destructor Documentation

4.165.2.1 StratmasTime::StratmasTime (const StratmasTime & *c*) [inline, protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.165.2.2 StratmasTime::StratmasTime (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.165.2.3 StratmasTime::StratmasTime (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.165.3 Member Function Documentation

4.165.3.1 ostream & StratmasTime::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.165.3.2 DataObject* StratmasTime::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.165.3.3 Time StratmasTime::getTime () const [inline, virtual]

Accessor.

Returns:

The current value.

Reimplemented from **DataObject** (p. 151).

4.165.3.4 DataObject & StratmasTime::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.165.3.5 void StratmasTime::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.165.3.6 void StratmasTime::setTime (Time *v*) [inline, virtual]

Mutator.

Parameters:

v The new value.

Reimplemented from **DataObject** (p. 153).

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

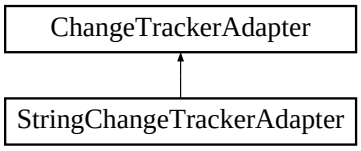
- DataObjectImpl.h
- DataObjectImpl.cpp

4.166 StringChangeTrackerAdapter Class Reference

The StringChangeTrackerAdapter keeps track of changes in **StratmasString**(p. 488) objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for StringChangeTrackerAdapter::



Public Member Functions

- **StringChangeTrackerAdapter** (**StratmasString** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 497) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **StratmasString** & **mObject**

  The adapted **DataObject**(p. 145).

- std::string **mLast**

  The last value written.

4.166.1 Detailed Description

The StringChangeTrackerAdapter keeps track of changes in **StratmasString**(p. 488) objects.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.166.2 Constructor & Destructor Documentation

4.166.2.1 StringChangeTrackerAdapter::StringChangeTrackerAdapter (StratmasString & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.166.3 Member Function Documentation

4.166.3.1 bool StringChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 497) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 497) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.166.3.2 ostream & StringChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.167 StrX Class Reference

This is a simple class for transcoding of XMLCh data to local code page for display.

`#include <StrX.h>`

Public Member Functions

- **StrX** (const XMLCh *const toTranscode)

  Constructor that transcodes an XMLCh string to a char string.

- ~**StrX** ()

  Destructor.

- const char * **str** () const

  Accessor for the string.

- bool **operator==** (const char *str)

  Comparsion operator for char arrays.

Private Attributes

- char **mStr** [**kBlock**]

  Buffer(p. 67) *for the string in char array form.*

- char * **mLongStr**

  Pointer to allocated memory if string is to large to fit in mStr.

Static Private Attributes

- const unsigned int **kBlock** = 256

  Default size of a block to transcode.

Friends

- std::ostream & **operator<<** (std::ostream &o, const **StrX** &s)

  For printing this string to an ostream.

- **Error** & **operator<<** (**Error** &e, const **StrX** &s)

  For printing this string to an **Error**(p. 205).

4.167.1 Detailed Description

This is a simple class for transcoding of XMLCh data to local code page for display.

Inspired by the StrX class in the DOMCount example in the Xerces-c distribution

Author:
Per Alexius

Date:Date
2006/09/12 11:54:20

4.167.2 Constructor & Destructor Documentation

4.167.2.1 StrX::StrX (const XMLCh *const *toTranscode*) [inline]

Constructor that transcodes an XMLCh string to a char string.

Parameters:
toTranscode The string to transcode

4.167.3 Member Function Documentation

4.167.3.1 bool StrX::operator== (const char * *str*) [inline]

Comparsion operator for char arrays.

Parameters:
str The string to compare to.

Returns:
true if the strings are lexicographically equal, false otherwise.

4.167.3.2 const char* StrX::str () const [inline]

Accessor for the string.

Returns:
The string as a char array

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

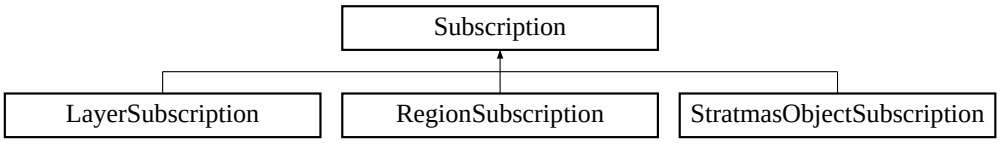
- StrX.h

4.168 Subscription Class Reference

Abstract base class for Subscriptions.

`#include <Subscription.h>`

Inheritance diagram for Subscription::



Public Member Functions

- **Subscription** (DOMElement *n, **Buffer** &buf)
- virtual ~**Subscription** ()

  Destructor.

- int **id** () const

  Accessor for the id of this Subscription.

- virtual void **getSubscribedData** (std::ostream &o)=0

  Writes an XML representation of the subscribed data to the provided stream.

Protected Attributes

- **Buffer** & **mBuf**

  Reference(p. 378) *to the* **Buffer**(p. 67).

- int **mId**

  The id of this Subscription.

4.168.1 Detailed Description

Abstract base class for Subscriptions.

All Subscriptions must implement the 'getSubscribedData()' method that fetches data from the **Buffer**(p. 67) mBuf and produces an XML representation of that data in accordance to the stratmasProtocol.xsd schema.

Author:
Per Alexius

Date:Date
2006/07/05 14:49:47

4.168.2 Constructor & Destructor Documentation

4.168.2.1 Subscription::Subscription (DOMElement * *n*, Buffer & *buf*)

Base class constructor. Reads the subscriptionId.

Parameters:

n The DOMElement to create the Subscription from.

buf The **Buffer**(p. 67) to fetch data from.

4.168.3 Member Function Documentation

4.168.3.1 virtual void Subscription::getSubscribedData (std::ostream & *o*) [pure virtual]

Writes an XML representation of the subscribed data to the provided stream.

Parameters:

o The stream to write to.

Implemented in **StratmasObjectSubscription** (p. 475), **LayerSubscription** (p. 275), and **RegionSubscription** (p. 390).

4.168.3.2 int Subscription::id () const [inline]

Accessor for the id of this Subscription.

Returns:

The id for this Subscription.

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

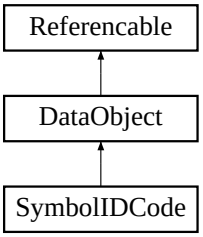
- Subscription.h
- Subscription.cpp

4.169 SymbolIDCode Class Reference

StratmasSymbolIDCode corresponds to the SymbolIDCode type in the Stratmas xml schema.

`#include <DataObjectImpl.h>`

Inheritance diagram for SymbolIDCode::



Public Member Functions

- **SymbolIDCode** (const **Reference** &scope, const DOMElement *n)

  Constructor that creates a **DataObject**(p. 145) *in the provided scope from the provided DOMElement.*

- **SymbolIDCode** (const **Reference** &ref, const **Type** &type)

  Constructor that creates a **DataObject**(p. 145) *of the specified* **Type**(p. 528) *with the provided* **Reference**(p. 378).

- std::string **getString** () const

  Accessor.

- void **setString** (const std::string &v)

  Mutator.

- **DataObject** & **operator**= (const **DataObject** &d)

  Assignment operator.

- **DataObject** * **clone** () const

  Creates a clone of this **DataObject**(p. 145).

- std::ostream & **bodyXML** (std::ostream &o, std::string indent) const

  Produces an XML representation of the body of this **DataObject**(p. 145) *according to the xml schemas.*

- void **print** (std::ostream &o, const std::string indent) const

  For debug purposes.

Protected Member Functions

- **SymbolIDCode** (const **SymbolIDCode** &c)

  Copy constructor.

Private Attributes

- std::string **mValue**

  The value.

4.169.1 Detailed Description

StratmasSymbolIDCode corresponds to the SymbolIDCode type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/03/27 09:43:40

4.169.2 Constructor & Destructor Documentation

4.169.2.1 SymbolIDCode::SymbolIDCode (const SymbolIDCode & *c*) [inline, protected]

Copy constructor.

Parameters:
c The **DataObject**(p. 145) to copy.

4.169.2.2 SymbolIDCode::SymbolIDCode (const Reference & *scope*, const DOMElement * *n*)

Constructor that creates a **DataObject**(p. 145) in the provided scope from the provided DOMElement.

Parameters:
scope A **Reference**(p. 378) the scope to create the **DataObject**(p. 145) in.

n The DOMElement to create this **DataObject**(p. 145) from.

4.169.2.3 SymbolIDCode::SymbolIDCode (const Reference & *ref*, const Type & *type*) [inline]

Constructor that creates a **DataObject**(p. 145) of the specified **Type**(p. 528) with the provided **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) to the **DataObject**(p. 145) tp be created.

type The **Type**(p. 528) of the **DataObject**(p. 145) to be created.

4.169.3 Member Function Documentation

4.169.3.1 ostream & SymbolIDCode::bodyXML (std::ostream & *o*, std::string *indent*) const [virtual]

Produces an XML representation of the body of this **DataObject**(p. 145) according to the xml schemas.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Returns:

The ostream with the XML representation written to it.

Implements **DataObject** (p. 149).

4.169.3.2 DataObject* SymbolIDCode::clone () const [inline, virtual]

Creates a clone of this **DataObject**(p. 145).

Returns:

A clone of this **DataObject**(p. 145).

Implements **DataObject** (p. 149).

4.169.3.3 std::string SymbolIDCode::getString () const [inline, virtual]

Accessor.

Returns:

The current value.

Reimplemented from **DataObject** (p. 150).

4.169.3.4 DataObject & SymbolIDCode::operator= (const DataObject & *d*) [virtual]

Assignment operator.

Parameters:

d The object to copy.

Returns:

The assigned object.

Reimplemented from **DataObject** (p. 151).

4.169.3.5 void SymbolIDCode::print (std::ostream & *o*, const std::string *indent*) const [virtual]

For debug purposes.

Parameters:

o The ostream to print to.

indent Intention string for readable output.

Reimplemented from **DataObject** (p. 152).

4.169.3.6 void SymbolIDCode::setString (const std::string & *v*) [inline, virtual]

Mutator.

Parameters:

v The new value.

Reimplemented from **DataObject** (p. 153).

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

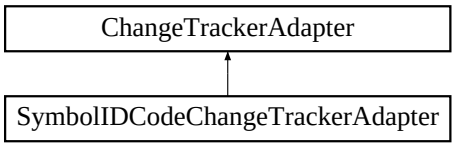
- DataObjectImpl.h
- DataObjectImpl.cpp

4.170 SymbolIDCodeChangeTrackerAdapter Class Reference

The SymbolIDCodeChangeTrackerAdapter keeps track of changes in StratmasSymbolIDCode objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for SymbolIDCodeChangeTrackerAdapter::



Public Member Functions

- **SymbolIDCodeChangeTrackerAdapter** (**SymbolIDCode** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 507) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **SymbolIDCode** & **mObject**

  The adapted **DataObject**(p. 145).

- std::string **mLast**

  The last value written.

4.170.1 Detailed Description

The SymbolIDCodeChangeTrackerAdapter keeps track of changes in StratmasSymbolIDCode objects.

Author:
Per Alexius

Date **Date**
2006/03/02 17:06:51

4.170.2 Constructor & Destructor Documentation

4.170.2.1 SymbolIDCodeChangeTrackerAdapter::SymbolIDCodeChangeTrackerAdapter (SymbolIDCode & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.170.3 Member Function Documentation

4.170.3.1 bool SymbolIDCodeChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 507) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 507) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.170.3.2 ostream & SymbolIDCodeChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

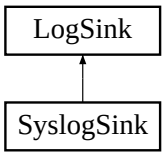
- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.171 SyslogSink Class Reference

This class implements **LogSink**(p. 289) using Posix 1003.1-2001 calls (i. e. syslog(3)).

`#include <SyslogSink.h>`

Inheritance diagram for SyslogSink::



Public Member Functions

- virtual void **sink** (const **LogMessage** *const message)

  syslog(3)'s the provieded message.

Static Public Member Functions

- **SyslogSink** * **createSyslogSink** (const std::string &ident)

  Creates a new Syslog sink, calls openlog(3) if this is the first object that will be created. If this is the first call, it will use the provided string as identification (typically the name of the program).

Static Private Attributes

- std::string **sIdent**
- bool **sSyslogOpened** = false

4.171.1 Detailed Description

This class implements **LogSink**(p. 289) using Posix 1003.1-2001 calls (i. e. syslog(3)).

Author:
Daniel Ahlin

Date **Date**
2006/07/25 14:52:00

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

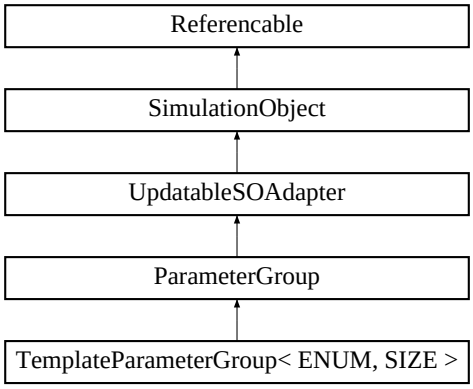
- SyslogSink.h
- SyslogSink.cpp

4.172 TemplateParameterGroup< ENUM, SIZE > Class Template Reference

Helper class which purpose is to facilitate creation of new ParameterGroups.

`#include <ParameterGroup.h>`

Inheritance diagram for TemplateParameterGroup< ENUM, SIZE >::



Public Member Functions

- **TemplateParameterGroup** (const **DataObject** &d, const **ParameterGroupEntry** *const groupEntries, int numGroupEntries, const **ParameterEntry** *const entries)
- virtual ~**TemplateParameterGroup** ()
- virtual void **prepareForSimulation** ()

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- void **replaceObject** (**DataObject** &newObject, int64_t initiator)

  Replaces the **SimulationObject**(p. 429) *with the same reference as the provided* **DataObject**(p. 145) *with a new* **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145).

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **reset** (const **DataObject** &d)

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- double **param** (ENUM i) const

  Accessor for parameters.

- std::ostream & **printMe** (std::ostream &o, std::string indent) const

Private Member Functions

- void **setDefault** ()

  Sets values to default.

- void **getDataFromDataObject** (const **DataObject** &d)

Private Attributes

- const **ParameterGroupEntry** *const **kGroupEntries**
- const int **kNumGroupEntries**
- const **ParameterEntry** *const **kEntries**
- std::map< std::string, **ParameterGroup** * > **mParameterGroups**
- double **mParameters** [SIZE+1]
- std::map< std::string, ENUM > **mNameToIndex**

4.172.1 Detailed Description

template<class ENUM, int SIZE> class TemplateParameterGroup< ENUM, SIZE >

Helper class which purpose is to facilitate creation of new ParameterGroups.

Author:
Per Alexius

Date:
2006/09/04 14:39:16

4.172.2 Constructor & Destructor Documentation

4.172.2.1 template<class ENUM, int SIZE> TemplateParameterGroup< ENUM, SIZE >::TemplateParameterGroup (const DataObject & *d*, const ParameterGroupEntry *const *groupEntries*, int *numGroupEntries*, const ParameterEntry *const *entries*) [inline]

Creates a parameter group from the provided **DataObject**(p. 145). The **ParameterEntry**(p. 313) list determines which parameters that are required for this to be a **ParameterGroup**(p. 314) of the desired 'type'

Parameters:

d The **DataObject**(p. 145) to create this **SimulationObject**(p. 429) from.

entries The **ParameterEntry**(p. 313) list determining which parameters that are required and types and default values.

4.172.2.2 template<class ENUM, int SIZE> virtual TemplateParameterGroup< ENUM, SIZE >::~TemplateParameterGroup () [inline, virtual]

Destructor.

4.172.3 Member Function Documentation

4.172.3.1 template<class ENUM, int SIZE> void TemplateParameterGroup< ENUM, SIZE >::addObject (DataObject & *toAdd*, int64_t *initiator*) [inline, virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:

toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 564).

4.172.3.2 template<class ENUM, int SIZE> void TemplateParameterGroup< ENUM, SIZE >::extract (Buffer & *b*) const [inline, virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:

b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.172.3.3 template<class ENUM, int SIZE> void TemplateParameterGroup< ENUM, SIZE >::modify (const DataObject & *d*) [inline, virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) containing the new value.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.172.3.4 template<class ENUM, int SIZE> double TemplateParameterGroup< ENUM, SIZE >::param (ENUM *i*) const [inline]

Accessor for parameters.

Parameters:

i The index of the parameter as specified in kEntries.

Returns:

The value of the given parameter.

4.172.3.5 template<class ENUM, int SIZE> virtual void TemplateParameterGroup< ENUM, SIZE >::prepareForSimulation () [inline, virtual]

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Implements **ParameterGroup** (p. 314).

4.172.3.6 template<class ENUM, int SIZE> void TemplateParameterGroup< ENUM, SIZE >::removeObject (const Reference & *toRemove*, int64_t *initiator*) [inline, virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:

toRemove The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.172.3.7 template<class ENUM, int SIZE> void TemplateParameterGroup< ENUM, SIZE >::replaceObject (DataObject & *newObject*, int64_t *initiator*) [inline, virtual]

Replaces the **SimulationObject**(p. 429) with the same reference as the provided **DataObject**(p. 145) with a new **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145).

Parameters:

newObject The **DataObject**(p. 145) to create the replacing object from.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.172.3.8 template<class ENUM, int SIZE> void TemplateParameterGroup< ENUM, SIZE >::reset (const DataObject & *d*) [inline, virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) to use as source for the reset.

Implements **SimulationObject** (p. 431).

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

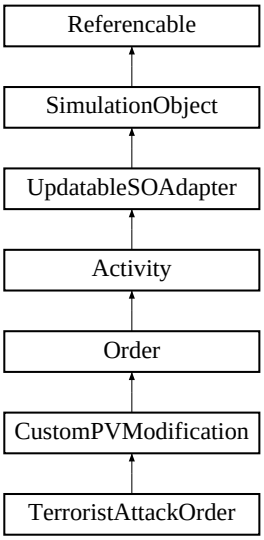
- ParameterGroup.h

4.173 TerroristAttackOrder Class Reference

The TerroristAttackOrder.

`#include <Activity.h>`

Inheritance diagram for TerroristAttackOrder::



Public Member Functions

- **TerroristAttackOrder** (const **DataObject** &d)

  Creates a TerroristAttackOrder object from the provided **DataObject**(p. 145).

- **Time actionTime** () const

  Accessor for the action time.

- bool **isActive** (**Time** t)

  Checks if this activity is active at time t.

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- void **perform** (**Element** *e, double fraction=1.0)

  Performs this **Activity**(p. 21).

Private Attributes

- **Time mActionTime**

  The time for the attack.

- bool **mTimeToPerform**

  True if the simulation time has passed the action time.

- bool **mAttackPerformed**

  True if the attack has been performed.

4.173.1 Detailed Description

The TerroristAttackOrder.

Author:
Per Alexius

Date:
2006/07/19 07:04:26

4.173.2 Constructor & Destructor Documentation

4.173.2.1 TerroristAttackOrder::TerroristAttackOrder (const DataObject & *d*)

Creates a TerroristAttackOrder object from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) from which to create this object.

4.173.3 Member Function Documentation

4.173.3.1 Time TerroristAttackOrder::actionTime () const [inline]

Accessor for the action time.

Returns:
The action time.

4.173.3.2 void TerroristAttackOrder::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Reimplemented from **CustomPVModification** (p. 142).

4.173.3.3 bool TerroristAttackOrder::isActive (Time *t*) [inline, virtual]

Checks if this activity is active at time t.

Parameters:
t The time for which to check.

Returns:
True if this activity is active at the specified time.

Reimplemented from **CustomPVModification** (p. 142).

4.173.3.4 void TerroristAttackOrder::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **CustomPVModification** (p. 142).

4.173.3.5 void TerroristAttackOrder::perform (Element * *e*, double *fraction* = 1.0) [virtual]

Performs this **Activity**(p. 21).

Parameters:
e The **Element**(p. 180) that should perform this **Activity**(p. 21).
fraction The fraction of the performers total capacity that this activity is performed with.

Reimplemented from **CustomPVModification** (p. 142).

4.173.3.6 void TerroristAttackOrder::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **CustomPVModification** (p. 143).

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

- Activity.h
- Activity.cpp

4.174 Time Class Reference

This class is used to represent timestamps and intervalls.

`#include <Time.h>`

Public Member Functions

- **Time** ()
  Default constructor.
- **Time** (const **Time** &t)
  Copy constructor.
- **Time** (int64_t d, int64_t h=0, int64_t m=0, int64_t s=0, int64_t ms=0)
  Constructor.
- int64_t **days** () const
  Returns the number of hours represented by this Time.
- double **hoursd** () const
  Returns the number of hours represented by this Time.
- int64_t **milliSeconds** () const
  Returns the number of milliseconds represented by this Time.
- void **addDays** (int ndays)
  Add a number of days.
- void **addHours** (int nhours)
  Add a number of hours.
- bool **isValid** () const
  Checks if this Time is a valid time.
- bool **operator**< (const **Time** &t) const
  Less-than operator.
- bool **operator**> (const **Time** &t) const
  Greater-than operator.
- bool **operator**<= (const **Time** &t) const
  Less-than-or-equal-to operator.
- bool **operator**>= (const **Time** &t) const
  Greater-than-or-equal-to operator.
- bool **operator**== (const **Time** &t) const
  Equality operator.

- bool **operator!**= (const **Time** &t) const

  Not-equal-to operator.

- **Time** & **operator**+= (const **Time** &t)

  Add an intervall to this Time.

- **Time operator**+ (const **Time** &t)

  Add two Time's.

- **Time** & **operator-**= (const **Time** &t)

  Subtract a Time from this Time.

- **Time operator-** (const **Time** &t)

  Subtract one Time from another.

Static Public Member Functions

- **Time maxTime** ()

  Returns the maximum time that Stratmas may represent.

- **Time minTime** ()

  Returns the minimum time that Stratmas may represent.

Private Attributes

- int64_t **mMilliSec**

  The number of seconds from the reference time.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Time** &t)

  For debugging purposes.

4.174.1 Detailed Description

This class is used to represent timestamps and intervalls.

The internal representation is a 64-bit integer holding the number of milliseconds, either from some reference time or in the intervall.

Author:
Per Alexius

Date **Date**
2006/09/20 09:21:01

4.174.2 Constructor & Destructor Documentation

4.174.2.1 Time::Time (int64_t *d*, int64_t *h* = 0, int64_t *m* = 0, int64_t *s* = 0, int64_t *ms* = 0) [inline]

Constructor.

Parameters:

d The number of days from the reference time or in the intervall.

h The number of hours from the reference time or in the intervall.

m The number of minutes from the reference time or in the intervall.

s The number of seconds from the reference time or in the intervall.

ms The number of milliseconds from the reference time or in the intervall.

4.174.3 Member Function Documentation

4.174.3.1 void Time::addDays (int *ndays*) [inline]

Add a number of days.

Parameters:

ndays The number of days to add.

4.174.3.2 void Time::addHours (int *nhours*) [inline]

Add a number of hours.

Parameters:

nhours The number of hours to add.

4.174.3.3 int64_t Time::days () const [inline]

Returns the number of hours represented by this Time.

Returns:

The number of hours

4.174.3.4 double Time::hoursd () const [inline]

Returns the number of hours represented by this Time.

Returns:

The number of hours

4.174.3.5 bool Time::isValid () const [inline]

Checks if this Time is a valid time.

Returns:
True if this time is valid, false otherwise.

4.174.3.6 Time Time::maxTime () [inline, static]

Returns the maximum time that Stratmas may represent.

Returns:
The maximum time that Stratmas may represent.

4.174.3.7 int64_t Time::milliSeconds () const [inline]

Returns the number of milliseconds represented by this Time.

Returns:
The number of milliseconds

4.174.3.8 Time Time::minTime () [inline, static]

Returns the minimum time that Stratmas may represent.

Returns:
The minimum time that Stratmas may represent.

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

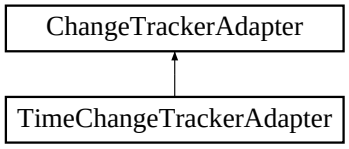
- Time.h

4.175 TimeChangeTrackerAdapter Class Reference

The TimeChangeTrackerAdapter keeps track of changes in **StratmasTime**(p. 492) objects.

`#include <ChangeTrackerAdapter.h>`

Inheritance diagram for TimeChangeTrackerAdapter::



Public Member Functions

- **TimeChangeTrackerAdapter** (**StratmasTime** &v)

  Creates a **ChangeTrackerAdapter**(p. 82) *for the provided* **DataObject**(p. 145).

- bool **changed** () const

  Checks if the **DataObject**(p. 145) *this adapter adapts has changed since the last call to the* **toXML()**(p. 521) *function.*

- std::ostream & **toXML** (std::ostream &o, std::string indent)

  Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Private Attributes

- **StratmasTime** & **mObject**

  The adapted **DataObject**(p. 145).

- **Time mLast**

  The last value written.

4.175.1 Detailed Description

The TimeChangeTrackerAdapter keeps track of changes in **StratmasTime**(p. 492) objects.

Author:
Per Alexius

Date:
2006/03/02 17:06:51

4.175.2 Constructor & Destructor Documentation

4.175.2.1 TimeChangeTrackerAdapter::TimeChangeTrackerAdapter (StratmasTime & *v*)

Creates a **ChangeTrackerAdapter**(p. 82) for the provided **DataObject**(p. 145).

Parameters:

v The **DataObject**(p. 145) to track changes for.

4.175.3 Member Function Documentation

4.175.3.1 bool TimeChangeTrackerAdapter::changed () const [virtual]

Checks if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 521) function.

Returns:

True if the **DataObject**(p. 145) this adapter adapts has changed since the last call to the **toXML()**(p. 521) function, false otherwise.

Implements **ChangeTrackerAdapter** (p. 83).

4.175.3.2 ostream & TimeChangeTrackerAdapter::toXML (std::ostream & *o*, std::string *indent*) [virtual]

Produces an XML representation of the changes in the object this adapter adapts according to the Stratmas xml schema. An indentation may be specified to increase readability.

Parameters:

o The ostream to write the XML representation to.

indent The whitespace indentation.

Returns:

The provided ostream with the XML representation written to it.

Implements **ChangeTrackerAdapter** (p. 83).

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

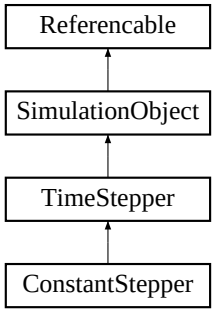
- ChangeTrackerAdapter.h
- ChangeTrackerAdapter.cpp

4.176 TimeStepper Class Reference

An abstract base class for all TimeSteppers.

`#include <TimeStepper.h>`

Inheritance diagram for TimeStepper::



Public Member Functions

- **TimeStepper** (const **DataObject** &d)
  Creates a TimeStepper from the provided **DataObject**(p. 145).
- virtual **Time dt** ()=0
  Get the length of the timestep.

4.176.1 Detailed Description

An abstract base class for all TimeSteppers.

Author:
Per Alexius

Date:
2006/03/06 14:23:13

4.176.2 Constructor & Destructor Documentation

4.176.2.1 TimeStepper::TimeStepper (const DataObject & *d*) [inline]

Creates a TimeStepper from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this TimeStepper from.

4.176.3 Member Function Documentation

4.176.3.1 virtual Time TimeStepper::dt () [pure virtual]

Get the length of the timestep.

Returns:
The length of the timestep.

Implemented in **ConstantStepper** (p. 123).

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

- TimeStepper.h

4.177 TranscoderWrapper Class Reference

Wraps an XMLTranscoder object to be used by the **StrX**(p. 498) and **XStr**(p. 603) classes.

`#include <StrX.h>`

Static Public Member Functions

- void **setEncoding** (const XMLCh *encoding, unsigned int maxCharSize)
  Sets the encoding to use when transcoding strings with the **StrX**(p. 498) *and* **XStr**(p. 603) *classes.*
- void **setEncoding** (const char *encoding, unsigned int maxCharSize)
  Sets the encoding to use when transcoding strings with the **StrX**(p. 498) *and* **XStr**(p. 603) *classes.*
- XMLTranscoder * **getTranscoder** ()
  Accessor for the transcoder object.
- unsigned int **getBlockSize** ()
  Accessor for the block size.
- unsigned int **getMaxCharSize** ()
  Accessor for the maximum char size of the current encoding.

Static Private Attributes

- const int **kBlockSize** = 1024 * 16
  Block size used internally by the parser. Should according to xercesc documentation be in the 4 to 64k range.
- unsigned int **sMaxCharSize**
- XMLTranscoder * **sTranscoder**
  Pointer to the transcoder object.

4.177.1 Detailed Description

Wraps an XMLTranscoder object to be used by the **StrX**(p. 498) and **XStr**(p. 603) classes.

Author:
Per Alexius

Date:
2006/09/12 11:54:20

4.177.2 Member Function Documentation

4.177.2.1 unsigned int TranscoderWrapper::getBlockSize () [inline, static]

Accessor for the block size.

Returns:
The block size.

4.177.2.2 unsigned int TranscoderWrapper::getMaxCharSize () [inline, static]

Accessor for the maximum char size of the current encoding.

Returns:
The maximum char size of the current encoding.

4.177.2.3 XMLTranscoder* TranscoderWrapper::getTranscoder () [inline, static]

Accessor for the transcoder object.

Returns:
The transcoder object.

4.177.2.4 void TranscoderWrapper::setEncoding (const char * *encoding*, unsigned int *maxCharSize*) [inline, static]

Sets the encoding to use when transcoding strings with the **StrX**(p. 498) and **XStr**(p. 603) classes.

Parameters:
encoding The name of the encoding, for example 'ISO-8859-1'.
maxCharSize The maximum size of a character in the given encoding.

4.177.2.5 void TranscoderWrapper::setEncoding (const XMLCh * *encoding*, unsigned int *maxCharSize*) [inline, static]

Sets the encoding to use when transcoding strings with the **StrX**(p. 498) and **XStr**(p. 603) classes.

Parameters:
encoding The name of the encoding, for example 'ISO-8859-1'.
maxCharSize The maximum size of a character in the given encoding.

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

- StrX.h
- XMLHandler.cpp

4.178 TSQueue< T > Class Template Reference

Threadsafe wrapper around the standard library queue.

`#include <TSQueue.h>`

Public Member Functions

- **TSQueue** ()
  Constructor.
- ~**TSQueue** ()
  Destructor.
- void **enqueue** (T t)
  Enqueue an element.
- T **dequeue** ()
  Dequeue an element or block if there is no element.
- unsigned int **size** () const
  Returns the number of elements in the queue.

Private Attributes

- boost::mutex **mLock**
  Semaphore for this queue.
- boost::condition **mEmpty**
  Condition variable for blocking on empty queue.
- std::queue< T > **mQ**
  The queue.

4.178.1 Detailed Description

template<class T> class TSQueue< T >

Threadsafe wrapper around the standard library queue.

If a thread tries to dequeue an element when the queue is empty, the thread will block until another thread enqueues an element. Can be used for message passing between threads.

Author:
Per Alexius

Date:
2006/07/03 14:18:23

4.178.2 Member Function Documentation

4.178.2.1 template<class T> T TSQueue< T >::dequeue () [inline]

Dequeue an element or block if there is no element.

Returns:
The dequeued element.

4.178.2.2 template<class T> void TSQueue< T >::enqueue (T *t*) [inline]

Enqueue an element.

Parameters:
t The element to be enqueued.

4.178.2.3 template<class T> unsigned int TSQueue< T >::size () const [inline]

Returns the number of elements in the queue.

Returns:
The number of elements in the queue

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

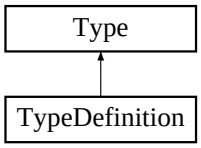
- TSQueue.h

4.179 Type Class Reference

The Type class represents a Type in the Stratmas xml schema.

`#include <Type.h>`

Inheritance diagram for Type::



Public Member Functions

- **Type** (**XSDContent** &content)

  Constructor that sets the creator **XSDContent**(p. 600).

- virtual ~**Type** ()

  Destructor.

- virtual const std::string & **getName** () const =0

  Accessor for the name of this Type.

- virtual const std::string & **getNamespace** () const =0

  Accessor for the namespace of this Type.

- virtual bool **abstract** () const =0

  Accessor for the abstract flag.

- const **Declaration** * **getSubElement** (const std::string &name) const

  Gets a sub element of this Type.

- const std::vector< const **Declaration** * > & **subElements** () const

  Accessor for the subelements vector.

- virtual bool **canSubstitute** (const **Type** &type) const =0

  Checks if this Type inherits from the provided Type.

- bool **canSubstitute** (const std::string &typeName) const

  Checks if this Type inherits from the Type with the specified name in the default namespace.

- bool **canSubstitute** (const std::string &typeName, const std::string &nameSpace) const

  Checks if this Type inherits from the Type with the specified name and namespace.

- bool **operator==** (const **Type** &t) const

  Equality operator. Two types are equal if they have identical name and namespace.

Protected Member Functions

- void **appendSubElement** (**Declaration** &dec)

  Appends a subelement to this Type.

- void **appendAttribute** (**TypeAttribute** &ta)

  Appends an attribute to this Type.

Protected Attributes

- **XSDContent** & **mXSDContent**

  The **XSDContent**(p. 600) *that this Type was created by.*

Private Attributes

- std::vector< const **Declaration** * > **mSubElements**

  The children of this type. The order of the elements in this vector is the same as in the sequence in the Stratmas XML schema.

- std::map< std::string, const **Declaration** * > **mSubElementsMap**

  Map(p. 292) *that maps child name to the child itself.*

- std::vector< const **TypeAttribute** * > **mAttributes**

  Vector containing the Attirbutes of this Type.

- std::map< std::string, const **TypeAttribute** * > **mAttributesMap**

  Map(p. 292) *that maps attribute name to the attribute itself.*

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Type** &t)

  For debugging purposes.

4.179.1 Detailed Description

The Type class represents a Type in the Stratmas xml schema.

Author:
Per Alexius

Date **Date**
2006/03/02 17:06:56

4.179.2 Member Function Documentation

4.179.2.1 virtual bool Type::abstract () const [pure virtual]

Accessor for the abstract flag.

Returns:
True if this Type is abstract, false otherwise.

Implemented in **TypeDefinition** (p. 537).

4.179.2.2 void Type::appendAttribute (TypeAttribute & *ta*) [protected]

Appends an attribute to this Type.

Parameters:
ta The attirbute to append.

4.179.2.3 void Type::appendSubElement (Declaration & *dec*) [protected]

Appends a subelement to this Type.

Parameters:
dec The **Declaration**(p. 160) to append.

4.179.2.4 bool Type::canSubstitute (const std::string & *typeName*, const std::string & *nameSpace*) const

Checks if this Type inherits from the Type with the specified name and namespace.

Parameters:
typeName The name of the Type to check inheritance from.
nameSpace The namespace of the Type to check inheritance from.

Returns:
True if this Type inherits from the provided Type.

4.179.2.5 bool Type::canSubstitute (const std::string & *typeName*) const

Checks if this Type inherits from the Type with the specified name in the default namespace.

Parameters:
typeName The name of the Type to check inheritance from.

Returns:
True if this Type inherits from the provided Type.

4.179.2.6 virtual bool Type::canSubstitute (const Type & *type*) const [pure virtual]

Checks if this Type inherits from the provided Type.

Parameters:
type The Type to check inheritance from.

Returns:
True if this Type inherits from the provided Type.

Implemented in **TypeDefinition** (p. 538).

4.179.2.7 virtual const std::string& Type::getName () const [pure virtual]

Accessor for the name of this Type.

Returns:
The name of this Type.

Implemented in **TypeDefinition** (p. 538).

4.179.2.8 virtual const std::string& Type::getNamespace () const [pure virtual]

Accessor for the namespace of this Type.

Returns:
The namespace of this Type.

Implemented in **TypeDefinition** (p. 538).

4.179.2.9 const Declaration * Type::getSubElement (const std::string & *name*) const

Gets a sub element of this Type.

Parameters:
name The name of the subelement to get.

Returns:
The sub element with the specified name or null of no such element was found.

4.179.2.10 bool Type::operator== (const Type & *t*) const [inline]

Equality operator. Two types are equal if they have identical name and namespace.

Parameters:
t The Type to check for equality.

Returns:
True if this Type and the provided Type are equal.

4.179.2.11 const std::vector<const Declaration*>& Type::subElements () const [inline]

Accessor for the subelements vector.

Returns:
The subelements vector.

4.179.3 Friends And Related Function Documentation

4.179.3.1 std::ostream& operator<< (std::ostream & *o*, const Type & *t*) [friend]

For debugging purposes.

Parameters:
o The stream to write to.

t The Type to print.

Returns:
The provided ostream with the Type written to it.

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

- Type.h
- Type.cpp

4.180 TypeAttribute Class Reference

The TypeAttribute class represents an attribute of a type in the Stratmas xml schema.

`#include <TypeAttribute.h>`

Public Member Functions

- **TypeAttribute** (XSAttributeUse &xsAttrUse)

  Constructs a TypeAttribute from the provided XSAttributeUse.

- const std::string & **getName** () const

  Accessor for the name.

- bool **required** () const

  Accessor for the required flag.

- int **constraintType** () const

  Accessor for the constraint type.

- const std::string & **constraintValue** () const

  Accessor for the constraint value.

Private Attributes

- std::string **mName**

  The name of the attribute.

- int **mConstraintType**

  The constraint type as specified by Xerces.

- bool **mRequired**

  Required flag.

- std::string **mConstraintValue**

  Fixed or default value if any.

Friends

- std::ostream & **operator**<< (std::ostream &o, const **TypeAttribute** &t)

  For debugging purposes.

4.180.1 Detailed Description

The TypeAttribute class represents an attribute of a type in the Stratmas xml schema.

Author:
Per Alexius

Date:
2006/05/24 12:32:11

4.180.2 Constructor & Destructor Documentation

4.180.2.1 TypeAttribute::TypeAttribute (XSAttributeUse & *xsAttrUse*)

Constructs a TypeAttribute from the provided XSAttributeUse.

Parameters:
xsAttrUse The XSAttributeUse to use for construction.

4.180.3 Member Function Documentation

4.180.3.1 int TypeAttribute::constraintType () const [inline]

Accessor for the constraint type.

Returns:
The constraint type.

4.180.3.2 const std::string& TypeAttribute::constraintValue () const [inline]

Accessor for the constraint value.

Returns:
The constraint value.

4.180.3.3 const std::string& TypeAttribute::getName () const [inline]

Accessor for the name.

Returns:
The name.

4.180.3.4 bool TypeAttribute::required () const [inline]

Accessor for the required flag.

Returns:
The state of the required flag.

4.180.4 Friends And Related Function Documentation

4.180.4.1 std::ostream& operator<< (std::ostream & *o*, const TypeAttribute & *t*) [friend]

For debugging purposes.

Parameters:

o The stream to write to.

t The TypeAttribute to print.

Returns:

The provided ostream with the TypeAttribute written to it.

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

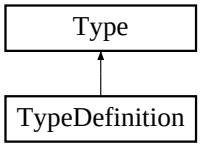
- TypeAttribute.h
- TypeAttribute.cpp

4.181 TypeDefinition Class Reference

The TypeDefinition class is the implementation of the **Type**(p. 528) interface.

`#include <TypeDefinition.h>`

Inheritance diagram for TypeDefinition::



Public Member Functions

- **TypeDefinition** (**XSDContent** &content, XSTypeDefinition &xsTypeDef)
  Creates a TypeDefinition.
- const std::string & **getName** () const
  Accessor for the name of this **Type**(p. 528).
- const std::string & **getNamespace** () const
  Accessor for the namespace of this **Type**(p. 528).
- bool **abstract** () const
  Accessor for the abstract flag.
- bool **canSubstitute** (const **Type** &type) const
  Checks if this **Type**(p. 528) *inherits from the provided* **Type**(p. 528).

Private Member Functions

- void **processSimpleTypeDefinition** (XSSimpleTypeDefinition &xsSimpleTypeDef)
  Handles simple type definitions.
- void **processComplexTypeDefinition** (XSComplexTypeDefinition &xsComplexTypeDef)
  Handles complex type definitions.
- void **processParticle** (XSParticle &xsParticle)
  Handles a single XSParticle.
- void **processParticles** (XSModelGroup &xsModelGroup)
  Handles a group of XSParticles.
- void **processAttributeUse** (XSAttributeUse &xsAttributeUse)
  Handles an XSAttributeuse.

Private Attributes

- XSTypeDefinition & **mXSTypeDefinition**

  The XSTypeDefinition that this TypeDefinition was creted from.

- std::string **mName**

  The name of this **Type**(p. 528).

- std::string **mNamespace**

  The namespace of this **Type**(p. 528).

- bool **mAbstract**

  Indicates if this **Type**(p. 528) *is abstract.*

4.181.1 Detailed Description

The TypeDefinition class is the implementation of the **Type**(p. 528) interface.

Author:
Per Alexius

Date Date
2006/05/24 12:32:11

4.181.2 Constructor & Destructor Documentation

4.181.2.1 TypeDefinition::TypeDefinition (XSDContent & *content*, XSTypeDefinition & *xsTypeDef*)

Creates a TypeDefinition.

Parameters:
content The **XSDContent**(p. 600) to create this TypeDefinition from.

xsTypeDef The XSTypeDefinition to create this TypeDefinition from.

4.181.3 Member Function Documentation

4.181.3.1 bool TypeDefinition::abstract () const [inline, virtual]

Accessor for the abstract flag.

Returns:
True if this **Type**(p. 528) is abstract, false otherwise.

Implements **Type** (p. 530).

4.181.3.2 bool TypeDefinition::canSubstitute (const Type & *type*) const [virtual]

Checks if this **Type**(p. 528) inherits from the provided **Type**(p. 528).

Parameters:
type The **Type**(p. 528) to check inheritance from.

Returns:
True if this **Type**(p. 528) inherits from the provided **Type**(p. 528).

Implements **Type** (p. 531).

4.181.3.3 const std::string& TypeDefinition::getName () const [inline, virtual]

Accessor for the name of this **Type**(p. 528).

Returns:
The name of this **Type**(p. 528).

Implements **Type** (p. 531).

4.181.3.4 const std::string& TypeDefinition::getNamespace () const [inline, virtual]

Accessor for the namespace of this **Type**(p. 528).

Returns:
The namespace of this **Type**(p. 528).

Implements **Type** (p. 531).

4.181.3.5 void TypeDefinition::processAttributeUse (XSAttributeUse & *xsAttributeUse*) [private]

Handles an XSAttributeuse.

Parameters:
xsAttributeUse The XSAttributeuse. to handle.

4.181.3.6 void TypeDefinition::processComplexTypeDefinition (XSComplexTypeDefinition & *xsComplexTypeDef*) [private]

Handles complex type definitions.

Parameters:
xsComplexTypeDef The XSComplexTypeDefinition to handle.

4.181.3.7 void TypeDefinition::processParticle (XSParticle & *xsParticle*) [private]

Handles a single XSParticle.

Parameters:
xsParticle The XSParticle to handle.

4.181.3.8 void TypeDefinition::processParticles (XSModelGroup & *xsModelGroup*) [private]

Handles a group of XSParticles.

Parameters:
xsModelGroup The XSModelGroup to handle.

4.181.3.9 void TypeDefinition::processSimpleTypeDefinition (XSSimpleTypeDefinition & *xsSimpleTypeDef*) [private]

Handles simple type definitions.

Parameters:
xsSimpleTypeDef The XSSimpleTypeDefinition to handle.

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

- TypeDefinition.h
- TypeDefinition.cpp

4.182 TypeFactory Class Reference

Factory for creating Types. The **XSDContent**(p. 600) Ccaches already created Types.

`#include <TypeFactory.h>`

Static Public Member Functions

- const **Type** & **getType** (const std::string &typeName)

  Creates a **Type**(p. 528) *object that corresponds to the xml schema type with the specified name.*

- const **Type** & **getType** (const std::string &typeName, const std::string &nameSpace)

  Creates a **Type**(p. 528) *object that corresponds to the xml schema type with the specified name and namespace.*

Static Private Attributes

- **XSDContent** ∗ **sXSDContent** = 0

  The **XSDContent**(p. 600) *to use for creating Types.*

4.182.1 Detailed Description

Factory for creating Types. The **XSDContent**(p. 600) Ccaches already created Types.

Author:
Per Alexius

Date:
2006/03/02 17:06:57

4.182.2 Member Function Documentation

4.182.2.1 const Type & TypeFactory::getType (const std::string & *typeName*, const std::string & *nameSpace*) [static]

Creates a **Type**(p. 528) object that corresponds to the xml schema type with the specified name and namespace.

Parameters:

typeName The name of the **Type**(p. 528).

nameSpace The namespace of the **Type**(p. 528).

Returns:
The **Type**(p. 528) with the specified name and namespace.

4.182.2.2 const Type & TypeFactory::getType (const std::string & *typeName*) [static]

Creates a **Type**(p. 528) object that corresponds to the xml schema type with the specified name.

Parameters:

typeName The name of the **Type**(p. 528).

Returns:

The **Type**(p. 528) with the specified name.

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

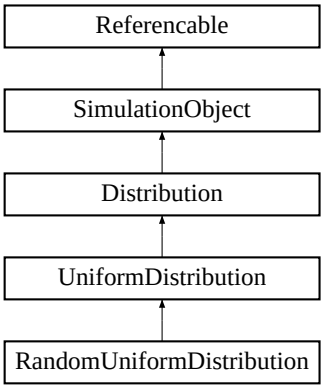
- TypeFactory.h
- TypeFactory.cpp

4.183 UniformDistribution Class Reference

A uniform distribution.

`#include <Distribution.h>`

Inheritance diagram for UniformDistribution::



Public Member Functions

- **UniformDistribution** (const **DataObject** &d)
  Creates a UniformDistribution from the provided **DataObject**(p. 145).
- double **f** (double x) const
  Gets the value of the distribution at distance x.
- void **extract** (**Buffer** &b) const
  Extracts data from this object to the **Buffer**(p. 67).

4.183.1 Detailed Description

A uniform distribution.

Author:
Per Alexius

Date:
2006/04/21 15:54:49

4.183.2 Constructor & Destructor Documentation

4.183.2.1 UniformDistribution::UniformDistribution (const DataObject & *d*) [inline]

Creates a UniformDistribution from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use for construction.

4.183.3 Member Function Documentation

4.183.3.1 void UniformDistribution::extract (Buffer & *b*) const [inline, virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:
b The **Buffer**(p. 67) to extract data to.

Implements **SimulationObject** (p. 430).

4.183.3.2 double UniformDistribution::f (double *x*) const [inline, virtual]

Gets the value of the distribution at distance x.

Parameters:
x The distance.

Returns:
The value of the distribution at distance x.

Implements **Distribution** (p. 175).

Reimplemented in **RandomUniformDistribution** (p. 374).

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

- Distribution.h

4.184 UniqueTime Class Reference

This is a helper class used to separate the different times that different passive clients are interested in receiving data for.

`#include <Engine.h>`

Public Member Functions

- **UniqueTime** (const **Time** &t)

  Creates a UniqueTime for the provided time.

- **UniqueTime** (const **UniqueTime** &t)

  Copy constructor.

- **Time time** () const

  Accessor for the time.

- bool **operator<** (const **UniqueTime** &t) const

  Less than operator.

Private Attributes

- int **mId**

  This UniqueTime's id.

- **Time mTime**

  This UniqueTime's time.

Static Private Attributes

- int **smNextId** = 0

  The next id to generate.

4.184.1 Detailed Description

This is a helper class used to separate the different times that different passive clients are interested in receiving data for.

Author:
Per Alexius

Date Date
2006/03/06 09:18:09

4.184.2 Constructor & Destructor Documentation

4.184.2.1 UniqueTime::UniqueTime (const Time & *t*) [inline]

Creates a UniqueTime for the provided time.

Parameters:
t The time.

4.184.2.2 UniqueTime::UniqueTime (const UniqueTime & *t*) [inline]

Copy constructor.

Parameters:
t The UniqueTime to copy.

4.184.3 Member Function Documentation

4.184.3.1 bool UniqueTime::operator< (const UniqueTime & *t*) const [inline]

Less than operator.

Parameters:
t The UniqueTime to compare to.

Returns:
True if this object is less than the provided object, false otherwise.

4.184.3.2 Time UniqueTime::time () const [inline]

Accessor for the time.

Returns:
The time.

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

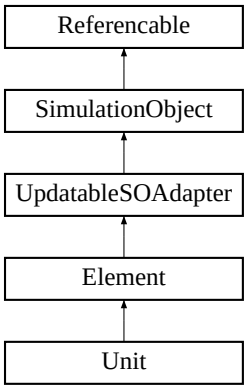
- Engine.h
- Engine.cpp

4.185 Unit Class Reference

This is the class that represents the simulation instance of a military unit.

`#include <Unit.h>`

Inheritance diagram for Unit::



Public Member Functions

- **Unit** (const **DataObject** &d)

  Creates a Unit object from the provided **DataObject**(p. 145).

- virtual ~**Unit** ()

  Destructor.

- void **prepareForSimulation** (**Grid** &grid, **Time** currentTime)

  Prepares this **SimulationObject**(p. 429) *for simulation.*

- void **extract** (**Buffer** &b) const

  Extracts data from this object to the **Buffer**(p. 67).

- void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

- void **reset** (const **DataObject** &d)

  Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

- void **addSubunit** (**Unit** *sub)

Adds a subunit to this unit.

- **Unit** * **parent** () const

  Returns the parent of this unit or null if it has no parent.

- const std::vector< **Unit** * > & **subunits** () const

  Accessor for the subunit vector.

- int **personnel** () const

  Accessor for the personnel.

- **Faction** & **faction** () const

  Returns the faction that this unit belongs to.

- int **casualties** () const

  Accessor for the casualties.

- double **strength** () const

  Accessor for the strength.

- double **initialStrength** () const

  Accessor for the initial strength.

- double **modifiedStrength** () const

  Accessor for the modified strength.

- **Time deployTime** () const

  Accessor for the deploy time.

- **Time departTime** () const

  Accessor for the depart time.

- bool **deployed** () const

  Accessor for the deployed flag.

- bool **departed** () const

  Accessor for the departed flag.

- int **color** () const

  Accessor for the color.

- bool **capable** () const

  Checks if this unit is capable of performing its activities.

- const **Shape** * **goal** () const

  Accessor for the goal.

- bool **present** () const

  Checks if this unit is currently present in the simulation, i.e. deployed and not departed.

- void **deploy** ()

  Deploy this unit.

- void **depart** ()

  Depart this unit.

- bool **untouchable** () const

  Returns true if this unit is untouchable.

- void **setup** (**Time** simTime)

  Sets up this unit before ti can act.

- void **act** (**Time** simTime)

  Perform this units tasks.

- void **setGoal** (const **Shape** &goal)

  Sets the goal for this unit.

- void **setVelocity** (double frac=0.5)

  Sets the velocity of this unit.

- void **move** ()

  Moves this unit with its current velocity straight towards its goal.

- bool **combat** ()

  Performs combat actions for this unit.

- bool **combatSituation** ()

  Checks if this unit is in a combat situation.

- bool **criticalInsurgentSituation** ()

  Checks if this unit is in a critical insurgent situation.

- bool **searching** ()

  Checks if this unit is searching for an ambushing unit.

- **PresenceObject** * **getPresence** (**PresenceObjectAllocator** &poa, int cellIndex, double fraction=1.0)

  Gets a **PresenceObject**(p. 338) *for this unit for the given cell.*

- bool **isHostileTowards** (const **Unit** &u) const

  Returns true if this unit is hostile towards the unit c.

- void **registerCombat** (**CombatGrid** &cg)

  Registers the effect of force on force combat, insurgent combat and ambush during this time step, e.g. modifies the personnel and casualties if necessary.

- double **registerInsurgentImpact** (int cell, double impact)

  registers the impact of insurgents in the specified cell. Does not change the personnel and casualties number since this is done in **registerCombat()**(p. 558)

- void **registerEnemy** (const **PresenceObject** &p)

  Registers a unit as an enemy of this unit.

- void **registerPotentialAmbush** (**PresenceObject** &victim, double myFraction)

  Registers a unit as an ambush victim of this unit if the ambush conditions are met.

- void **setLocation** (const **Shape** &area)

  Sets the location of this unit.

- void **registerSpotter** (const **PresenceObject** &p)

  Registers the unit in the given **PresenceObject**(p. 338) *as a potential 'spotter' of this unit.*

- bool **isSpotted** ()

  Determines if this unit is spotted by any of its potential spotters.

- void **kill** (**Unit** &killer)

  Kills this unit and spreads damage evenly across all overlapped cells.

Private Member Functions

- **Order** * **setAllocatedOrder** (**Order** *order)

  Helper for handling memory allocation for temporarily created orders that must survive beyond the scope of a single function.

- **Order** * **setOrder** (**Time** simTime)

  Sets the order of this unit.

- void **setModifiedStrength** ()

  Calculates and sets the modified strength for this unit based on a time step of one day (24 h).

- void **setUpSearchAndDestroy** (**Time** simTime)

  Finds out which of an ambushed unit's ambushers it will find the next timestep.

- **Order** * **createRetreatOrder** ()

  Creates a **RetreatOrder**(p. 394).

- void **exposeForAttack** (double modifiedStrength, **Unit** &attacker)

  Exposes this unit for an attack by another unit.

- void **exposeForAmbush** (**Unit** &u, int cell, double damage)

  Exposes this unit for an ambush. Also registers the damage suffered by the ambushing unit.

- bool **attacker** () const

  Checks if this unit is an attacker.

- bool **defender** () const

  Checks if this unit is an defender.

- double **attackDefendFactor** () const

  Gets the combat value modifier based on if we're attacker, defender or neither.

- void **recover** ()

  Increases the personnel of the unit due to recovering. According to Thorssell a unit recovers personnel corresponding to one unit of strength per day.

Private Attributes

- **Unit** * **mParent**

  The parent of this unit.

- std::string **mSymbolIDCode**

  The symbol id code.

- std::vector< **Unit** * > **mSubunits**

  A vector with this unit's subunits.

- const **Reference** * **mAffiliation**

  The affiliation of this unit.

- int **mPersonnel**

  The personnel of this unit.

- double **mPersonnelRest**

  Decimal accumulative personnel rest (recovered).

- int **mCasualties**

  The current number of casualties.

- double **mCasualtyRest**

  Decimal accumulative casualty rest.

- double **mInitialPersonnel**

  The initial personnel of this unit.

- double **mStrengthFactor**

  The strength factor.

- double **mAttackFactor**

  The attack factor.

- double **mDefenseFactor**

  The defense factor.

- double **mAttritionCoefficient**

  The attrition coefficient.

- double **mWithdrawThreshold**

The withdraw threshold.

- double **mVelocity**

  The current velocity of this unit.

- double **mMaxVelocity**

  The maximum velocity of this unit.

- double **mSqueeze**

  Used to compensate for current latitude.

- **Time mDeployTime**

  This unit's deploy time.

- **Time mDepartTime**

  This unit's depart time.

- bool **mDeployed**

  Indicates if this unit has been deployed.

- bool **mDeparted**

  Indicates if this unit has departed.

- bool **mMoving**

  Indicates if this unit is currently moving.

- **Shape** * **mGoal**

  The current goal of this unit.

- std::map< **Unit** *, **EnemyRecord** > **mEnemyRecords**

  Maps a unit to the **EnemyRecord**(p. 187) *for that unit.*

- std::map< int, **AmbushRecord** > **mAmbushRecords**

  Maps a cell index to the AmbudhRecord for that cell.

- std::map< **Unit** *, **EnemyRecord** > **mAmbushExposure**

  Maps a unit to the **EnemyRecord**(p. 187) *holding the damage caused by ambushs by that unit.*

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

  Cells where we were damaged by insurgents during the last time step.

- std::map< **Unit** *, double > **mSpotters**

  Stores potential spotters and their 'presence'.

- bool **mSearchWithoutFind**

  True if we were ambushed during the last time step.

- double **mCurrentEnemyStrengthSum**

  The sum of the strenth of enemies overlapping our area.

- double **mInsurgentRelatedCasualties**

  Damage caused by insurgents during the last time step.

- double **mModifiedStrength**

  The modified strength.

- std::vector< **Activity** * > **mActivities**

  A vector containing this unit's activities.

- **Order** * **mCurrentOrder**

  The order currently executed by this unit.

- **Order** * **mAllocatedOrder**

  Holds the address to the latest created order so it can be freed.

- double **mAngle**

  This unit's current face angle.

- **Grid** * **mGrid**

  The **Grid**(p. 227) *in which this unit lives.*

- int **mColor**

  The color of this unit (blue, red etc).

Friends

- std::ostream & **operator**<< (std::ostream &o, const **Unit** &u)

  For debugging purposes.

4.185.1 Detailed Description

This is the class that represents the simulation instance of a military unit.

Author:
Per Alexius

Date **Date**
2006/04/21 15:54:52

4.185.2 Constructor & Destructor Documentation

4.185.2.1 Unit::Unit (const DataObject & *d*)

Creates a Unit object from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create the unit from.

4.185.3 Member Function Documentation

4.185.3.1 void Unit::act (Time *simTime*)

Perform this units tasks.

Parameters:
simTime The current simulation time.

4.185.3.2 void Unit::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:
toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.
initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 564).

4.185.3.3 void Unit::addSubunit (Unit * *sub*) [inline]

Adds a subunit to this unit.

Parameters:
sub The subunit to add.

4.185.3.4 double Unit::attackDefendFactor () const [private]

Gets the combat value modifier based on if we're attacker, defender or neither.

Returns:
The attack defend combat value modifier.

4.185.3.5 bool Unit::attacker () const [private]

Checks if this unit is an attacker.

Returns:
True if this unit is an attacker, false otherwise.

4.185.3.6 bool Unit::capable () const [inline]

Checks if this unit is capable of performing its activities.

A unit that has lost more that mWithdrawThreshold percent of its total force strength is no longer considered capable.

Returns:
True if this unit is capable, fals otherwise.

4.185.3.7 int Unit::casualties () const [inline]

Accessor for the casualties.

Returns:
The casualties.

4.185.3.8 int Unit::color () const [inline]

Accessor for the color.

Returns:
The color of this unit.

4.185.3.9 bool Unit::combat ()

Performs combat actions for this unit.

Exposes each recorded enemy for an attack of size that is proportional to the strength of that unit compared to other ememies.

Returns:
True if we're stll in a combat situation, false otherwise.

4.185.3.10 bool Unit::combatSituation ()

Checks if this unit is in a combat situation.

Returns:
True if this unit will have to fight (or retreat).

4.185.3.11 Order * Unit::createRetreatOrder () [private]

Creates a **RetreatOrder**(p. 394).

Returns:
The newly created **RetreatOrder**(p. 394).

4.185.3.12 bool Unit::criticalInsurgentSituation ()

Checks if this unit is in a critical insurgent situation.

Critical means that the unit has lost more than one per thousand of its personnel during the last day due to insurgent activities.

Returns:
True if the insurgent situation is critical.

4.185.3.13 bool Unit::defender () const [private]

Checks if this unit is an defender.

Returns:
True if this unit is an defender, false otherwise.

4.185.3.14 bool Unit::departed () const [inline]

Accessor for the departed flag.

Returns:
The status of the departed flag.

4.185.3.15 Time Unit::departTime () const [inline]

Accessor for the depart time.

Returns:
The depart time.

4.185.3.16 bool Unit::deployed () const [inline]

Accessor for the deployed flag.

Returns:
The status of the deployed flag.

4.185.3.17 Time Unit::deployTime () const [inline]

Accessor for the deploy time.

Returns:
The deploy time.

4.185.3.18 void Unit::exposeForAmbush (Unit & *u*, int *cell*, double *damage*) [private]

Exposes this unit for an ambush. Also registers the damage suffered by the ambushing unit.

Parameters:
u The ambushing unit.

cell The cell in which this ambush is performed.

damage The damage caused to the **ambushing** unit, e.g. 5% of its personnel in this cell. The damage suffered by the ambushed unit is 4 times that number.

4.185.3.19 void Unit::exposeForAttack (double *modStr*, Unit & *attacker*) [private]

Exposes this unit for an attack by another unit.

This function implements the damage model proposed by Thorssell, where the magnitude of the damage is highly dependent of the quote between the attacker's and the defender's combat values. Notice that - despite the name of the function - the unit for which this function is called is not necessarily a defender.

Parameters:

modStr The modified strength (the part that should be used to combat this unit) of the attacker..

attacker The attacking unit.

4.185.3.20 void Unit::extract (Buffer & *b*) const [virtual]

Extracts data from this object to the **Buffer**(p. 67).

Parameters:

b The **Buffer**(p. 67) to extract data to.

Reimplemented from **Element** (p. 182).

4.185.3.21 Faction & Unit::faction () const

Returns the faction that this unit belongs to.

Returns:

The faction that this unit belongs to.

4.185.3.22 PresenceObject * Unit::getPresence (PresenceObjectAllocator & *poa*, int *cellIndex*, double *fraction* = 1.0)

Gets a **PresenceObject**(p. 338) for this unit for the given cell.

Parameters:

poa The **PresenceObjectAllocator**(p. 341) to use for allocation.

cellIndex The index of the cell the **PresenceObject**(p. 338) should refer to.

fraction The fraction of this unit that should be placed in this cell according to the deployment distribution of the unit. Notice that this is not necessarily the actual fraction since it may differ if this unit is in a critical insurgent situation and should thus focus its strength to the cells where insurgent related damage occured during the previous time step.

Returns:

A **PresenceObject**(p. 338) or null if this unit does not have any personnel.

4.185.3.23 const Shape* Unit::goal () const [inline]

Accessor for the goal.

Returns:
The goal.

4.185.3.24 double Unit::initialStrength () const [inline]

Accessor for the initial strength.

Returns:
The initial strength.

4.185.3.25 bool Unit::isHostileTowards (const Unit & *u*) const

Returns true if this unit is hostile towards the unit c.

Parameters:
u The unit to check stance against.

Returns:
True if this unit is hostile towards the unit c, false otherwise.

4.185.3.26 bool Unit::isSpotted ()

Determines if this unit is spotted by any of its potential spotters.

A spotted unit is killed immediately, according to the model.

Returns:
True if we're spotted, false otherwise.

4.185.3.27 double Unit::modifiedStrength () const [inline]

Accessor for the modified strength.

Returns:
The modified strength.

4.185.3.28 void Unit::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) containing the new value.

Reimplemented from **Element** (p. 182).

4.185.3.29 Unit* Unit::parent () const [inline]

Returns the parent of this unit or null if it has no parent.

Returns:
The parent of this unit or null if it has no parent.

4.185.3.30 int Unit::personnel () const [inline]

Accessor for the personnel.

Returns:
The personnel.

4.185.3.31 void Unit::prepareForSimulation (Grid & *grid*, Time *currentTime*)

Prepares this **SimulationObject**(p. 429) for simulation.

Should be called after creation and reset and before the simulation starts.

Parameters:
grid The **Grid**(p. 227).
currentTime The current simulation time.

4.185.3.32 bool Unit::present () const [inline, virtual]

Checks if this unit is currently present in the simulation, i.e. deployed and not departed.

Returns:
True if this unit is present, false otherwise.

Implements **Element** (p. 182).

4.185.3.33 void Unit::registerCombat (CombatGrid & *cg*)

Registers the effect of force on force combat, insurgent combat and ambush during this time step, e.g. modifies the personnel and casualties if necessary.

Parameters:
cg The **CombatGrid**(p. 101) to register casualties to.

4.185.3.34 void Unit::registerEnemy (const PresenceObject & *p*)

Registers a unit as an enemy of this unit.

Enemies will get attacked when **act()**(p. 553) is called the next time provided that this unit is capable.

Parameters:
p The **PresenceObject**(p. 338) for the victim.

4.185.3.35 double Unit::registerInsurgentImpact (int *cell*, double *impact*)

registers the impact of insurgents in the specified cell. Does not change the personnel and casualties number since this is done in **registerCombat()**(p. 558)

Parameters:

cell The index of the cell.

impact The magnitude of * the impact in number of casualties.

Returns:

The number of casualties.

4.185.3.36 void Unit::registerPotentialAmbush (PresenceObject & *victim*, double *myFraction*)

Registers a unit as an ambush victim of this unit if the ambush conditions are met.

The ambush conditions are: If there are more than kMinAmbushPerKm2 personnel per square kilometer in this cell the probability for an ambush is p = **personnel**(p. 558) in this cell / 1000.

Parameters:

victim The **PresenceObject**(p. 338) for the potential victim.

myFraction The fraction of this unit that is located in the cell refered to by the **PresenceObject**(p. 338).

4.185.3.37 void Unit::registerSpotter (const PresenceObject & *p*)

Registers the unit in the given **PresenceObject**(p. 338) as a potential 'spotter' of this unit.

Used in order to fulfill the TerroristAttack order behavior. Notice that a unit may only be spotted during the same timestep as the attack is (or isn't) performed.

Parameters:

p The **PresenceObject**(p. 338) containing the potential spotting unit and how large part of that unit that is located in the cell the **PresenceObject**(p. 338) refers to.

4.185.3.38 void Unit::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:

toRemove The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the update.

Reimplemented from **UpdatableSOAdapter** (p. 565).

4.185.3.39 void Unit::reset (const DataObject & *d*) [virtual]

Resets this object to the state it would have had if it was created from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to use as source for the reset.

Reimplemented from **Element** (p. 183).

4.185.3.40 bool Unit::searching ()

Checks if this unit is searching for an ambushing unit.

Returns:
True if this unit is searcing for an ambushing unit.

4.185.3.41 Order * Unit::setAllocatedOrder (Order * *order*) [private]

Helper for handling memory allocation for temporarily created orders that must survive beyond the scope of a single function.

The only purpose is to deallocate the last allocated order and store a pointer to the newly allocated order so that it may be freed at the next invocation of this function.

Parameters:
order The newly allocated order.

Returns:
The same order as the input order.

4.185.3.42 void Unit::setGoal (const Shape & *goal*)

Sets the goal for this unit.

Parameters:
goal The new goal

4.185.3.43 void Unit::setLocation (const Shape & *newLocation*)

Sets the location of this unit.

Parameters:
newLocation The new location.

4.185.3.44 Order * Unit::setOrder (Time *simTime*) [private]

Sets the order of this unit.

Parameters:
simTime The current simulation time.

4.185.3.45 void Unit::setup (Time *simTime*)

Sets up this unit before ti can act.

Must set up all units before any of them can act since we have to know for example which order each unit has.

Parameters:
simTime The current simulation time.

4.185.3.46 void Unit::setUpSearchAndDestroy (Time *simTime*) [private]

Finds out which of an ambushed unit's ambushers it will find the next timestep.

If it finds any of them - create an **AttackOrder**(p. 55) for itself and a defend order for each of the found units. Then it will be regular combat during the next timestep.

4.185.3.47 double Unit::strength () const [inline]

Accessor for the strength.

Returns:
The strength.

4.185.3.48 const std::vector<Unit*>& Unit::subunits () const [inline]

Accessor for the subunit vector.

Returns:
The subunit vector.

4.185.3.49 bool Unit::untouchable () const

Returns true if this unit is untouchable.

An untouchable state allows the unit to move without beeing attacked by enemies. This is currently used when a unit has retreated and then should go to the location of its closest superior and for the Ambush order.

Returns:
True if this unit is untouchable.

4.185.4 Friends And Related Function Documentation

4.185.4.1 std::ostream& operator<< (std::ostream & *o*, const Unit & *u*) [friend]

For debugging purposes.

Parameters:

o The stream to write to.

u The Unit to print.

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

- Unit.h
- Unit.cpp

4.186 UpdatableSOAdapter Class Reference

Conveniance class that provides some default update behavior for SimulationObjects.

`#include <UpdatableSOAdapter.h>`

Inheritance diagram for UpdatableSOAdapter::

Public Member Functions

- virtual void **update** (const **Update** &u)

  Update(p. 567) *dispatcher function.*

- virtual void **addObject** (**DataObject** &toAdd, int64_t initiator)

  Adds the **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145) *to this object.*

- virtual void **removeObject** (const **Reference** &toRemove, int64_t initiator)

  Removes the **SimulationObject**(p. 429) *referenced by the provided* **Reference**(p. 378) *from this object.*

- virtual void **replaceObject** (**DataObject** &newObject, int64_t initiator)

  Replaces the **SimulationObject**(p. 429) *with the same reference as the provided* **DataObject**(p. 145) *with a new* **SimulationObject**(p. 429) *created from the provided* **DataObject**(p. 145).

- virtual void **modify** (const **DataObject** &d)

  Modifies this object with data from the provided **DataObject**(p. 145).

Protected Member Functions

- **UpdatableSOAdapter** (const **Reference** &ref)

  Creates a UpdatableSOAdapter with the specified **Reference**(p. 378).

- **UpdatableSOAdapter** (const **Referencable** &ref)

  Creates a UpdatableSOAdapter from the specified **Referencable**(p. 375).

- **UpdatableSOAdapter** (const **DataObject** &d)

  Creates a UpdatableSOAdapter from the specified **DataObject**(p. 145).

4.186.1 Detailed Description

Conveniance class that provides some default update behavior for SimulationObjects.

Author:
Per Alexius

Date:
2006/07/03 14:18:23

4.186.2 Constructor & Destructor Documentation

4.186.2.1 UpdatableSOAdapter::UpdatableSOAdapter (const Reference & *ref*) [inline, protected]

Creates a UpdatableSOAdapter with the specified **Reference**(p. 378).

Parameters:
ref The **Reference**(p. 378) for the object to be created.

4.186.2.2 UpdatableSOAdapter::UpdatableSOAdapter (const Referencable & *ref*) [inline, protected]

Creates a UpdatableSOAdapter from the specified **Referencable**(p. 375).

Parameters:
ref A **Referencable**(p. 375) for the object to be created.

4.186.2.3 UpdatableSOAdapter::UpdatableSOAdapter (const DataObject & *d*) [inline, protected]

Creates a UpdatableSOAdapter from the specified **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.186.3 Member Function Documentation

4.186.3.1 void UpdatableSOAdapter::addObject (DataObject & *toAdd*, int64_t *initiator*) [virtual]

Adds the **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145) to this object.

Parameters:
toAdd The **DataObject**(p. 145) to create the new **SimulationObject**(p. 429) from.
initiator The id of the initiator of the update.

Reimplemented in **Order** (p. 309), **CustomPVModification** (p. 141), **AgencyTeam** (p. 36), **CustomAgencyTeam** (p. 137), **Faction** (p. 214), **TemplateParameterGroup< ENUM, SIZE >** (p. 511), **Scenario** (p. 398), **Simulation** (p. 426), **Unit** (p. 553), **TemplateParameterGroup< eInsurgentModelP, eNumInsurgentModelP >** (p. 511), **TemplateParameterGroup< NOTYPE, 0 >** (p. 511), and **TemplateParameterGroup< eFoodModelP, eNumFoodModelP >** (p. 511).

4.186.3.2 void UpdatableSOAdapter::modify (const DataObject & *d*) [virtual]

Modifies this object with data from the provided **DataObject**(p. 145).

Parameters:

d The **DataObject**(p. 145) containing the new value.

Reimplemented in **Activity** (p. 23), **Order** (p. 311), **CustomPVModification** (p. 142), **TerroristAttackOrder** (p. 515), **DefendOrder** (p. 167), **AmbushOrder** (p. 43), **AgencyTeam** (p. 37), **CustomAgencyTeam** (p. 137), **Element** (p. 182), **Faction** (p. 215), **TemplateParameterGroup< ENUM, SIZE >** (p. 511), **Scenario** (p. 399), **Simulation** (p. 427), **Unit** (p. 557), **TemplateParameterGroup< eInsurgentModelP, eNumInsurgentModelP >** (p. 511), **TemplateParameterGroup< NOTYPE, 0 >** (p. 511), and **TemplateParameterGroup< eFoodModelP, eNumFoodModelP >** (p. 511).

4.186.3.3 void UpdatableSOAdapter::removeObject (const Reference & *toRemove*, int64_t *initiator*) [virtual]

Removes the **SimulationObject**(p. 429) referenced by the provided **Reference**(p. 378) from this object.

Parameters:

toRemove The **Reference**(p. 378) to the object to remove.

initiator The id of the initiator of the update.

Reimplemented in **Order** (p. 311), **CustomPVModification** (p. 143), **AgencyTeam** (p. 38), **CustomAgencyTeam** (p. 137), **Faction** (p. 215), **TemplateParameterGroup< ENUM, SIZE >** (p. 512), **Scenario** (p. 399), **Simulation** (p. 427), **Unit** (p. 559), **TemplateParameterGroup< eInsurgentModelP, eNumInsurgentModelP >** (p. 512), **TemplateParameterGroup< NOTYPE, 0 >** (p. 512), and **TemplateParameterGroup< eFoodModelP, eNumFoodModelP >** (p. 512).

4.186.3.4 void UpdatableSOAdapter::replaceObject (DataObject & *newObject*, int64_t *initiator*) [virtual]

Replaces the **SimulationObject**(p. 429) with the same reference as the provided **DataObject**(p. 145) with a new **SimulationObject**(p. 429) created from the provided **DataObject**(p. 145).

Parameters:

newObject The **DataObject**(p. 145) to create the replacing object from.

initiator The id of the initiator of the update.

Reimplemented in **Order** (p. 311), **Element** (p. 183), **Faction** (p. 215), **TemplateParameterGroup< ENUM, SIZE >** (p. 512), **TemplateParameterGroup< eInsurgentModelP, eNumInsurgentModelP >** (p. 512), **TemplateParameterGroup< NOTYPE, 0 >** (p. 512), and **TemplateParameterGroup< eFoodModelP, eNumFoodModelP >** (p. 512).

4.186.3.5 void UpdatableSOAdapter::update (const Update & *u*) [virtual]

Update(p. 567) dispatcher function.

Parameters:

u The **Update**(p. 567) to update this object with.

Implements **SimulationObject** (p. 431).

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

- UpdatableSOAdapter.h
- UpdatableSOAdapter.cpp

4.187 Update Class Reference

Class representing an update sent by the client.

`#include <Update.h>`

Public Types

- enum **eType** { **eAdd**, **eRemove**, **eReplace**, **eModify** }

  Enumeration for Update types.

Public Member Functions

- **Update** (const DOMElement &n, int64_t initiator)

  Creates an Update from the provided DOMElement.

- ~**Update** ()

  The Update is responsible for deallocation of the **DataObject**(p. 145) *if and only if it is of type 'eModify'.*

- int64_t **getInitiator** () const

  Accessor for the initiator of this update.

- const **Reference** & **getReference** () const

  Accessor for the **Reference**(p. 378) *to the object the update refers to.*

- const **Reference** & **getTargetRef** () const

  Gets the reference to the object that should be notified of the update.

- **DataObject** * **getObject** () const

  Accessor for the object the update refers to.

- int **getType** () const

  Accessor for the type of this Update.

- const char * **getTypeAsString** () const

  Gets the type of this Update as a string.

Static Private Member Functions

- const **Reference** * **findReferenceToClosestComplexParent** (const **Reference** &ref)

  Finds the closest parent to the **DataObject**(p. 145) *pointed out by the provided* **Reference**(p. 378) *that is a* **ComplexDataObject**(p. 110).

Private Attributes

- int64_t **mInitiator**
  The id of the initiator of the update.
- const **Reference** * **mReference**
  Reference(p. 378) *to the object to be updated.*
- const **Reference** * **mTarget**
  Reference(p. 378) *to the object that should be notified of the update.*
- **DataObject** * **mObject**
  The **DataObject**(p. 145) *containing the update information except for remove updates that does not need such information.*
- int **mType**
  The type of Update according to the eType enumeration.

4.187.1 Detailed Description

Class representing an update sent by the client.

Author:
Per Alexius

Date **Date**
2006/05/24 12:32:12

4.187.2 Constructor & Destructor Documentation

4.187.2.1 Update::Update (const DOMElement & *n*, int64_t *initiator*)

Creates an Update from the provided DOMElement.

Parameters:
n The DOMElement to create this Update from.
initiator The id of the initiator of the Update.

4.187.3 Member Function Documentation

4.187.3.1 const Reference * Update::findReferenceToClosestComplexParent (const Reference & *ref*) [static, private]

Finds the closest parent to the **DataObject**(p. 145) pointed out by the provided **Reference**(p. 378) that is a **ComplexDataObject**(p. 110).

Parameters:
ref The **Reference**(p. 378) to the object to find the closest complex parent for.

Returns:
A **Reference**(p. 378) to the closest complex parent or null if no such parent was found.

4.187.3.2 int64_t Update::getInitiator () const [inline]

Accessor for the initiator of this update.

Returns:
The initiator of this object.

4.187.3.3 DataObject* Update::getObject () const [inline]

Accessor for the object the update refers to.

Returns:
The object the update refers to.

4.187.3.4 const Reference& Update::getReference () const [inline]

Accessor for the **Reference**(p. 378) to the object the update refers to.

Returns:
The **Reference**(p. 378) to the object the update refers to.

4.187.3.5 const Reference& Update::getTargetRef () const [inline]

Gets the reference to the object that should be notified of the update.

Returns:
The reference to the object that should be notified of the update.

4.187.3.6 int Update::getType () const [inline]

Accessor for the type of this Update.

Returns:
The type of this Update as specified in the eType enumeration.

4.187.3.7 const char * Update::getTypeAsString () const

Gets the type of this Update as a string.

Returns:
The type of this Update as a string as specified in the schema.

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

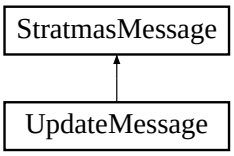
- Update.h
- Update.cpp

4.188 UpdateMessage Class Reference

Class representing the UpdateMessage.

`#include <StratmasMessage.h>`

Inheritance diagram for UpdateMessage::



Public Member Functions

- **UpdateMessage** (**Buffer** &b, **ChangeTrackerAdapter** &c, bool r)

  Constructor.

- **Time validForTime** () const

  Accessor for the simulation time for which the data in the last produced XML representation is valid.

- void **addSubscription** (**Subscription** *s)

  Adds a **Subscription**(p. 500) *to this message.*

- void **toXML** (std::ostream &o) const

  Produces the XML representation of this message.

Private Attributes

- **Buffer** & **mBuf**

  The **Buffer**(p. 67) *to fetch data from.*

- **ChangeTrackerAdapter** & **mChangeTracker**

  The ChangeTracker.

- bool **mRegisteredForUpdates**

  True if the client receiving this message is registered for updates.

- std::vector< **Subscription** * > **mSubscriptions**

  Vector containing pointers to all Subscriptions which data should be sent with this message.

- **Time mValidForTime**

  The simulation time for which the data in the last produced XML representation is valid.

4.188.1 Detailed Description

Class representing the UpdateMessage.

Author:
Per Alexius

Date:
2006/03/06 14:23:12

4.188.2 Constructor & Destructor Documentation

4.188.2.1 UpdateMessage::UpdateMessage (Buffer & *b*, ChangeTrackerAdapter & *c*, bool *r*) [inline]

Constructor.

Parameters:
- ***b*** The **Buffer**(p. 67).
- ***c*** The **ChangeTrackerAdapter**(p. 82).
- ***r*** Indicates if the client is registered for updates.

4.188.3 Member Function Documentation

4.188.3.1 void UpdateMessage::addSubscription (Subscription * *s*) [inline]

Adds a **Subscription**(p. 500) to this message.

Parameters:
- ***s*** The **Subscription**(p. 500) to be added

4.188.3.2 void UpdateMessage::toXML (std::ostream & *o*) const [virtual]

Produces the XML representation of this message.

Parameters:
- ***o*** The stream to which the message is written

Implements **StratmasMessage** (p. 473).

4.188.3.3 Time UpdateMessage::validForTime () const [inline]

Accessor for the simulation time for which the data in the last produced XML representation is valid.

Returns:
The simulation time for which the data in the last produced XML representation is valid.

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

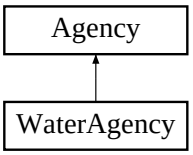
- StratmasMessage.h
- StratmasMessage.cpp

4.189 WaterAgency Class Reference

Class containing functionality for controlling WaterAgencyTeams.

`#include <Agency.h>`

Inheritance diagram for WaterAgency::



Public Member Functions

- **WaterAgency** (const std::vector< **AgencyTeam** * > &teams, **Grid** &g)

  Constructor.

Private Member Functions

- bool **severeProblem** ()

  Determines if we have a severe resource problem. If we do - then the weights for the clustering algorithm is set.

4.189.1 Detailed Description

Class containing functionality for controlling WaterAgencyTeams.

Author:
Per Alexius

Date:
2006/10/02 16:01:25

4.189.2 Constructor & Destructor Documentation

4.189.2.1 WaterAgency::WaterAgency (const std::vector< AgencyTeam * > & *teams*, Grid & *g*) [inline]

Constructor.

Parameters:
teams A vector containing this Agency's teams.

g A reference to the **Grid**(p. 227).

4.189.3 Member Function Documentation

4.189.3.1 bool WaterAgency::severeProblem () [private, virtual]

Determines if we have a severe resource problem. If we do - then the weights for the clustering algorithm is set.

Returns:
True if we have a severe water problem, false otherwise.

Implements **Agency** (p. 28).

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

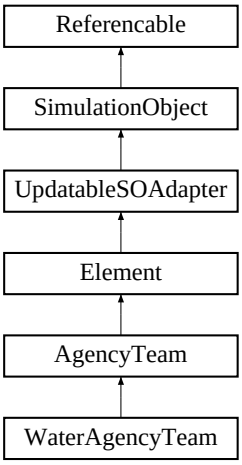
- Agency.h
- Agency.cpp

4.190 WaterAgencyTeam Class Reference

Class representing a WaterAgencyTeam.

`#include <AgencyTeam.h>`

Inheritance diagram for WaterAgencyTeam::



Public Member Functions

- **WaterAgencyTeam** (const **DataObject** &d)

  Constructor that creates an **AgencyTeam**(p. 32) *from the provided* **DataObject**(p. 145).

- double **calculateNeed** ()

  Calculates water need among the population in this team's area of influence.

- void **act** (**Time** now)

  Distribute water in this team's area of influence according to its capacity.

Private Attributes

- double **mRepairCapacity**

  Fraction of the agency's total capacity that this team has.

4.190.1 Detailed Description

Class representing a WaterAgencyTeam.

Author:
Per Alexius

Date:
2006/10/10 09:35:59

4.190.2 Constructor & Destructor Documentation

4.190.2.1 WaterAgencyTeam::WaterAgencyTeam (const DataObject & *d*) [inline]

Constructor that creates an **AgencyTeam**(p. 32) from the provided **DataObject**(p. 145).

Parameters:
d The **DataObject**(p. 145) to create this object from.

4.190.3 Member Function Documentation

4.190.3.1 void WaterAgencyTeam::act (Time *now*) [virtual]

Distribute water in this team's area of influence according to its capacity.

Parameters:
now The current simulation time.

Implements **AgencyTeam** (p. 35).

4.190.3.2 double WaterAgencyTeam::calculateNeed () [virtual]

Calculates water need among the population in this team's area of influence.

Water need is calculated as follows: For all cells in this team's area of influence with population $p > 0.5$ persons - add $f * p$ (where f is the fraction of the population that does not have water) to the total need.

Returns:
The need for water in this team's area of influence represented as persons without water.

Reimplemented from **AgencyTeam** (p. 36).

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

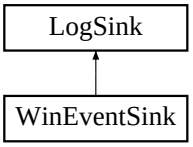
- AgencyTeam.h
- AgencyTeam.cpp

4.191 WinEventSink Class Reference

This class implements **LogSink**(p. 289) using Windows events.

`#include <WinEventSink.h>`

Inheritance diagram for WinEventSink::



Public Member Functions

- virtual void **sink** (const **LogMessage** *const message)

  Creates a windows INFORMATION event and posts it.

Static Public Member Functions

- **WinEventSink** * **createWinEventSink** (const std::string &sourceName)

  Creates a new Syslog sink, calls RegisterEventSource if this is the first object that will be created. If this is the first call, it will use the provided string as identification (typically the name of the program).

Static Private Attributes

- std::string **sSourceName**
- std::string **sServerName**
- HANDLE **sHandle** = 0
- bool **sEventSourcedRegistered** = false

4.191.1 Detailed Description

This class implements **LogSink**(p. 289) using Windows events.

Author:
Daniel Ahlin

Date **Date**
2006/07/25 14:52:00

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

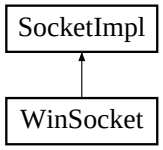
- WinEventSink.h
- WinEventSink.cpp

4.192 WinSocket Class Reference

C++ wrapper around a Windows socket.

`#include <WinSocket.h>`

Inheritance diagram for WinSocket::



Public Member Functions

- **WinSocket** ()
  Constructor.
- virtual ~**WinSocket** ()
  Destructor.
- virtual bool **create** ()
  Creates the underlying socket.
- virtual bool **bind** (const char *host, int port)
  Binds this socket to the provided address and port.
- virtual bool **listen** () const
  Listens to connections.
- virtual bool **accept** (**Socket** &newSock) const
  Accepts a connection.
- virtual bool **connect** (const std::string host, const int port)
  Connects to the specified port on the specified host.
- virtual bool **close** ()
  Closes the socket.
- virtual bool **send** (const void *msg, unsigned int len) const
  Sends data over the socket.
- virtual int **recv** (void *msg, unsigned int len) const
  Receives data from the socket.
- virtual int **recvf** (void *msg, int len) const
  Receives an exact amount of data from the socket.
- virtual void **set_non_blocking** (const bool b)

Set the O_NONBLOCK flag.

- virtual bool **valid** () const
  Checks if this socket is valid.

- virtual std::string **address** () const
  Returns a string representation of the address of this socket.

Static Public Member Functions

- bool **initWinSocketLibrary** ()

Protected Attributes

- SOCKET **mSock**
  The Windows socket object.

- sockaddr_in **mAddr**
  The name assigned to this socket.

Static Protected Attributes

- WSADATA **sWSAData**
  Library init data:.

- bool **sWSADataInitialized** = WinSocket::initWinSocketLibrary()
  Library init success.

4.192.1 Detailed Description

C++ wrapper around a Windows socket.

Author:
Daniel Ahlin

Date:Date
2006/07/03 14:18:24

4.192.2 Member Function Documentation

4.192.2.1 bool WinSocket::accept (Socket & *newSock*) const [virtual]

Accepts a connection.

Parameters:
newSock On return - the socket from which the connection was accepted.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.192.2.2 std::string WinSocket::address () const [virtual]

Returns a string representation of the address of this socket.

Returns:
A string representation of the address of this socket.

Implements **SocketImpl** (p. 435).

4.192.2.3 bool WinSocket::bind (const char * *host*, int *port*) [virtual]

Binds this socket to the provided address and port.

Parameters:
host The name of the host or null if INADDR_ANY should be used.
port The port.

Returns:
True if the socket was successfully bound, false otherwise.

Implements **SocketImpl** (p. 435).

4.192.2.4 bool WinSocket::close () [virtual]

Closes the socket.

Returns:
True on success.

Implements **SocketImpl** (p. 435).

4.192.2.5 bool WinSocket::connect (const std::string *host*, const int *port*) [virtual]

Connects to the specified port on the specified host.

Parameters:
host The host to connect to.
port The port to connect to.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.192.2.6 bool WinSocket::create () [virtual]

Creates the underlying socket.

Returns:
True if the socket was successfully created, false otherwise.

Implements **SocketImpl** (p. 435).

4.192.2.7 bool WinSocket::listen () const [virtual]

Listens to connections.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.192.2.8 int WinSocket::recv (void * *msg*, unsigned int *len*) const [virtual]

Receives data from the socket.

Parameters:
msg On return - the data received.

len The maximum length in bytes of the data to receive.

Returns:
True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.192.2.9 int WinSocket::recvf (void * *msg*, int *len*) const [virtual]

Receives an exact amount of data from the socket.

Parameters:
msg On return - the data received.

len The number of bytes to receive.

Returns:
The total number of bytes read.

Implements **SocketImpl** (p. 435).

4.192.2.10 bool WinSocket::send (const void * *msg*, unsigned int *len*) const [virtual]

Sends data over the socket.

Parameters:

msg The data to send.

len The length in bytes of the data to be sent.

Returns:

True if all is ok, false otherwise.

Implements **SocketImpl** (p. 435).

4.192.2.11 void WinSocket::set_non_blocking (const bool *b*) [virtual]

Set the O_NONBLOCK flag.

Parameters:

b New value of the O_NONBLOCK flag.

Implements **SocketImpl** (p. 435).

4.192.2.12 bool WinSocket::valid () const [virtual]

Checks if this socket is valid.

Returns:

True if this socket is valid, false otherwise.

Implements **SocketImpl** (p. 435).

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

- WinSocket.h
- WinSocket.cpp

4.193 XMLHandler Class Reference

This class handles the extraction of data from StratmasMessages.

`#include <XMLHandler.h>`

Public Member Functions

- **XMLHandler** (**Buffer** &buf, std::string ns, std::string schemaLocation, int64_t id)
  Initializes the parser and error reporter to be used.
- ~**XMLHandler** ()
  Destructor.
- **DataObject** * **takeOverSimulation** ()
  Removes and returns the simulation **DataObject**(p. 145).
- std::vector< **Update** * > **takeOverUpdates** ()
  Removes and returns the updates vector.
- bool **sessionBigEndian** () const
  Accessor for the client byte order indicator.
- bool **registeredForUpdatesFlag** () const
  Accessor for the flag indicating if the client is registered for updates.
- const std::string & **lastType** () const
  Accessor for the type of the last message handled.
- int **numberOfTimesteps** () const
  Accessor for the number of timesteps in the last handled StepMessage.
- bool **detachedStep** () const
  Accessor for the detached step flag.
- void **getSubscriptions** (**UpdateMessage** &um) const
  Gets data from the currently held subscriptions and puts it into the provided **UpdateMessage**(p. 570).
- void **getGridBasedSubscriptions** (**UpdateMessage** &um) const
  Gets data from the currently held subscriptions that has to do with the grid, e.g. Layer and **Region**(p. 386) *subscriptions and puts it into the provided* **UpdateMessage**(p. 570).
- int **handle** (const std::string &xml)
  Parses and extracts data from the provided xml document.
- void **createSubscription** (DOMElement &n)
  Creates a new subscription from the provided DOMElement.
- void **eraseSubscriptions** ()
  Deletes all subscriptions held by this XMLHandler.

Private Member Functions

- void **addSubscription** (**Subscription** *sub)

  Helper for adding a new **Subscription**(p. 500). *Performs some error handling.*

- void **handleConnectMessage** (DOMElement &n)

  Parses and extracts data from a ConnectMessage.

- void **handleRegisterForUpdatesMessage** (DOMElement &n)

  Parses and extracts data from a RegisterForUpdatesMessage.

- void **handleStepMessage** (DOMElement &n)

  Parses and extracts data from a StepMessage.

- void **handleServerUpdateMessage** (DOMElement &n)

  Parses and extracts data from a ServerUpdateMessage.

- void **handleSubscriptionMessage** (DOMElement &n)

  Parses and extracts data from a SubscriptionMessage.

- void **handleSetPropertyMessage** (DOMElement &n)

  Parses and extracts data from a SetPropertyMessage.

Private Attributes

- int64_t **mId**

  Id of the **Session**(p. 408) *this XMLHandler belongs to.*

- XMLEntityResolver * **mpEntityResolver**

  Resolver for parser resource requests.

- **DataObject** * **mSimulation**

  Pointer to the simulation **DataObject**(p. 145).

- **PVInitValueSet** * **mPVInitValueSet**

  The **PVInitValueSet**(p. 366).

- **Buffer** & **mBuf**

  Reference(p. 378) *to the* **Buffer**(p. 67) *object.*

- bool **mSessionBigEndian**

  Indicates the byte order of the client sending the messages to be handled.

- bool **mRegisteredForUpdates**

  Indicates if the client sending the messages is registered for updates.

- std::map< int, **Subscription** * > **mSubscriptions**

  Maps subscription id to the actual subscription. Notice that subscriptions are not stored anyplace else than in the XMLHandler.

- int **mNumberOfTimesteps**

  The number of timesteps to take before returning any data (extracted from the last handled StepMessage).

- bool **mDetachedStep**

  True if the last handled StepMessage was a 'detached' step message.

- std::vector< **Update** * > **mUpdates**

  A vector containing the Updates from the last message.

- std::string **mLastType**

  Type(p. 528) *of the last* **StratmasMessage**(p. 472) *handled.*

- XercesDOMParser * **mParser**

  The parser used to parse incoming messages.

- **ParserErrorReporter** * **mErrorReporter**

  The error reporter used during parsing.

4.193.1 Detailed Description

This class handles the extraction of data from StratmasMessages.

Author:
Per Alexius

Date **Date**
2007/01/24 13:13:26

4.193.2 Constructor & Destructor Documentation

4.193.2.1 XMLHandler::XMLHandler (Buffer & *buf*, std::string *ns*, std::string *schemaLocation*, int64_t *id*)

Initializes the parser and error reporter to be used.

Parameters:
buf A reference to the BufferObject.

ns The namespace part of the schemaLocation attribute.

schemaLocation Points out the schema to use for validation.

id The id of the **Session**(p. 408) this XMLHandler belongs to.

4.193.3 Member Function Documentation

4.193.3.1 void XMLHandler::addSubscription (Subscription * *sub*) [private]

Helper for adding a new **Subscription**(p. 500). Performs some error handling.

Parameters:
sub A pointer to the **Subscription**(p. 500) to add.

4.193.3.2 void XMLHandler::createSubscription (DOMElement & *n*)

Creates a new subscription from the provided DOMElement.

Parameters:
n The DOMElement containing data on the **Subscription**(p. 500) to be created.

4.193.3.3 bool XMLHandler::detachedStep () const [inline]

Accessor for the detached step flag.

Returns:
The status of the detached step flag.

4.193.3.4 void XMLHandler::getGridBasedSubscriptions (UpdateMessage & *um*) const

Gets data from the currently held subscriptions that has to do with the grid, e.g. Layer and **Region**(p. 386) subscriptions and puts it into the provided **UpdateMessage**(p. 570).

Parameters:
um The **UpdateMessage**(p. 570) into which to put the subscriptions.

4.193.3.5 void XMLHandler::getSubscriptions (UpdateMessage & *um*) const

Gets data from the currently held subscriptions and puts it into the provided **UpdateMessage**(p. 570).

Parameters:
um The **UpdateMessage**(p. 570) into which to put the subscriptions.

4.193.3.6 int XMLHandler::handle (const std::string & *xml*)

Parses and extracts data from the provided xml document.

Parameters:
xml The xml message to handle.

Returns:
The type of the message handled.

4.193.3.7 void XMLHandler::handleConnectMessage (DOMElement & *n*) [private]

Parses and extracts data from a ConnectMessage.

Parameters:
n The DOMElement containing the message to be handled.

4.193.3.8 void XMLHandler::handleRegisterForUpdatesMessage (DOMElement & *n*) [private]

Parses and extracts data from a RegisterForUpdatesMessage.

Parameters:
n The DOMElement containing the message to be handled.

4.193.3.9 void XMLHandler::handleServerUpdateMessage (DOMElement & *n*) [private]

Parses and extracts data from a ServerUpdateMessage.

Parameters:
n The DOMElement containing the message to be handled.

4.193.3.10 void XMLHandler::handleSetPropertyMessage (DOMElement & *n*) [private]

Parses and extracts data from a SetPropertyMessage.

Parameters:
n The DOMElement containing the message to be handled.

4.193.3.11 void XMLHandler::handleStepMessage (DOMElement & *n*) [private]

Parses and extracts data from a StepMessage.

Parameters:
n The DOMElement containing the message to be handled.

4.193.3.12 void XMLHandler::handleSubscriptionMessage (DOMElement & *n*) [private]

Parses and extracts data from a SubscriptionMessage.

Parameters:
n The DOMElement containing the message to be handled.

4.193.3.13 const std::string& XMLHandler::lastType () const [inline]

Accessor for the type of the last message handled.

Returns:
The type of the last message handled.

4.193.3.14 int XMLHandler::numberOfTimesteps () const [inline]

Accessor for the number of timesteps in the last handled StepMessage.

Returns:
The number of timesteps in the last handled StepMessage.

4.193.3.15 bool XMLHandler::registeredForUpdatesFlag () const [inline]

Accessor for the flag indicating if the client is registered for updates.

Returns:
True if the client whose messages we handle is registered for updates.

4.193.3.16 bool XMLHandler::sessionBigEndian () const [inline]

Accessor for the client byte order indicator.

Returns:
True if the client whose messages we handle has big endian byte order.

4.193.3.17 DataObject* XMLHandler::takeOverSimulation () [inline]

Removes and returns the simulation **DataObject**(p. 145).

Returns:
The simulation **DataObject**(p. 145).

4.193.3.18 std::vector<Update*> XMLHandler::takeOverUpdates () [inline]

Removes and returns the updates vector.

Returns:
The updates vector.

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

- XMLHandler.h
- XMLHandler.cpp

4.194 XMLHelper Class Reference

This class contains various static functions for handling xml related tasks.

`#include <XMLHelper.h>`

Static Public Member Functions

- std::string **nodeTypeToString** (int i)

  Maps node types to their names.

- std::string & **removeNamespace** (std::string &s)

  Removes the namespace from a string.

- std::string **encodeSpecialCharacters** (const std::string &s)

  Encodes XML special characters.

- std::string **encodeURLSpecialCharacters** (const std::string &s)

  Encodes XML special characters.

- void **timeToDateTime** (std::ostream &o, **Time** time)

  Prints a **Time**(p. 516) *object to the provided stream in XML Schema dateTime format. Always assumes UTC.*

- **Time dateTimeToTime** (const std::string &dateTime)

  Converts the given XML Schema dateTime string to a **Time**(p. 516) *object.*

- std::ostream & **base64Print** (const int8_t *toEncode, int nBytesToEncode, std::ostream &o)

  Base64 encodes the provided byte array.

- template<class T> std::ostream & **base64Print** (const T *const toEncode, int numElements, bool swapByteOrder, std::ostream &o)

  Base64 encodes data and werites it to the provided stream.

- const XMLCh * **getXMLChString** (const DOMElement &n, const char *tag)

  Gets an XMLCh string from the first subelement of the provided DOMElement that has the tag 'tag'.

- int **getIntAttribute** (const DOMElement &n, const char *tag)

  Gets an int representation of the attribute named 'tag' of the provided DOMElement.

- double **getDoubleAttribute** (const DOMElement &n, const char *tag)

  Gets an double representation of the attribute named 'tag' of the provided DOMElement.

- std::string **getStringAttribute** (const DOMElement &n, const char *tag)

  Gets a string representation of the attribute named 'tag' of the provided DOMElement.

- std::string **getTypeAttribute** (const DOMElement &n)

  Convenience function for getting the xsi:type attribute and stripping the leading namespace from it.

- bool **getBool** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a bool representation its content.

- double **getDouble** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a double representation its content.

- int **getInt** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int representation its content.

- int64_t **getLongInt** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int64_t representation its content.

- **Shape** * **getShape** (const DOMElement &n, const char *tag, const **Reference** &scope)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a **Shape**(p. 412) *representation its content.*

- void **getString** (const DOMElement &n, const char *tag, std::string &outStr)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a string representation its content.

- **Time getTime** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a **Time**(p. 516) *representation of its contents.*

- bool **getElementBoolValue** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a bool representation of its value subelement.

- int **getElementIntValue** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int representation of its value subelement.

- int64_t **getElementLongIntValue** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int64_t representation of its value subelement.

- double **getElementDoubleValue** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a double representation of its value subelement.

- void **getElementStringValue** (const DOMElement &n, const char *tag, std::string &out-Str)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a string representation of its value subelement.

- **Time getElementTimestampValue** (const DOMElement &n, const char *tag)

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a **Time**(p. 516) *representation of its value subelement.*

- **Shape** * **getShape** (const DOMElement &n, const **Reference** &scope)

  Gets a **Shape**(p. 412) *representation of the provided DOMElement.*

- DOMElement * **getFirstChildByTag** (const DOMElement &n, const char *tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag.

- DOMElement * **getFirstChildByTag** (const DOMElement &n, const std::string &tag)

  Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag.

- void **getChildElementsByTag** (const DOMElement &n, const char *tag, std::vector< DOMElement * > &ioV)

  Finds all subelements of the provided DOMElement that has a tag that matches the specified tag.

- void **getChildElementsByTag** (const DOMElement &n, const std::string &tag, std::vector< DOMElement * > &ioV)

  Finds all subelements of the provided DOMElement that has a tag that matches the specified tag.

Private Member Functions

- **XMLHelper** ()

4.194.1 Detailed Description

This class contains various static functions for handling xml related tasks.

Author:
Per Alexius

Date:
2007/01/24 13:13:27

4.194.2 Constructor & Destructor Documentation

4.194.2.1 XMLHelper::XMLHelper () [inline, private]

Private default constructor since this class should not be instantiated.

4.194.3 Member Function Documentation

4.194.3.1 template<class T> ostream & XMLHelper::base64Print (const T *const *toEncode*, int *numElements*, bool *swapByteOrder*, std::ostream & *o*) [inline, static]

Base64 encodes data and werites it to the provided stream.

Also handles swapping of byte order.

Parameters:

toEncode The data to encode.

numElements Number of elements in the provided data array.

swapByteOrder Should be set to true if the byte order should be swapped.

o The stream to write to.

Returns:

The provided stream with the base64 string written to it.

4.194.3.2 ostream & XMLHelper::base64Print (const int8_t * *toEncode*, int *nBytesToEncode*, std::ostream & *o*) [inline, static]

Base64 encodes the provided byte array.

Parameters:

toEncode The data to encode.

nBytesToEncode Number of bytes in the provided data array.

o The stream to write to.

Returns:

The provided stream with the base64 string written to it.

4.194.3.3 Time XMLHelper::dateTimeToTime (const std::string & *dateTime*) [inline, static]

Converts the given XML Schema dateTime string to a **Time**(p. 516) object.

Parameters:

dateTime The dateTime string.

Returns:

The **Time**(p. 516) object.

4.194.3.4 string XMLHelper::encodeSpecialCharacters (const std::string & *s*) [inline, static]

Encodes XML special characters.

Parameters:

s The string in which to encode the special characters.

Returns:

A new string with special characters encoded.

4.194.3.5 string XMLHelper::encodeURLSpecialCharacters (const std::string & *s*) [inline, static]

Encodes XML special characters.

Parameters:

s The string in which to encode the special characters.

Returns:

A new string with special characters encoded.

4.194.3.6 bool XMLHelper::getBool (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a bool representation its content.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The bool representation of the subelement's content.

4.194.3.7 void XMLHelper::getChildElementsByTag (const DOMElement & *n*, const std::string & *tag*, std::vector< DOMElement * > & *ioV*) [inline, static]

Finds all subelements of the provided DOMElement that has a tag that matches the specified tag.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

ioV A vector that on return contains all subelements with matching tag.

4.194.3.8 void XMLHelper::getChildElementsByTag (const DOMElement & *n*, const char * *tag*, std::vector< DOMElement * > & *ioV*) [inline, static]

Finds all subelements of the provided DOMElement that has a tag that matches the specified tag.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

ioV A vector that on return contains all subelements with matching tag.

4.194.3.9 double XMLHelper::getDouble (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a double representation its content.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The double representation of the subelement's content.

4.194.3.10 double XMLHelper::getDoubleAttribute (const DOMElement & *n*, const char * *tag*) [inline, static]

Gets an double representation of the attribute named 'tag' of the provided DOMElement.

Parameters:

n The DOMElement to get the attribute from.

tag The name of the attribute.

Returns:

The double representation of the attribute.

4.194.3.11 bool XMLHelper::getElementBoolValue (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a bool representation of its value subelement.

Used to extract data from ValueType descendants.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The bool representation of the value element's content.

4.194.3.12 double XMLHelper::getElementDoubleValue (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a double representation of its value subelement.

Used to extract data from ValueType descendants.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:
The double representation of the value element's content.

4.194.3.13 int XMLHelper::getElementIntValue (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int representation of its value subelement.

Used to extract data from ValueType descendants.

Parameters:
n The parent DOMElement.

tag The tag of the subelement.

Returns:
The int representation of the value element's content.

4.194.3.14 int64_t XMLHelper::getElementLongIntValue (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int64_t representation of its value subelement.

Used to extract data from ValueType descendants.

Parameters:
n The parent DOMElement.

tag The tag of the subelement.

Returns:
The int64_t representation of the value element's content.

4.194.3.15 void XMLHelper::getElementStringValue (const DOMElement & *n*, const char * *tag*, std::string & *outStr*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a string representation of its value subelement.

Used to extract data from ValueType descendants.

Parameters:
n The parent DOMElement.

tag The tag of the subelement.

outStr The string representation of the value element's content.

4.194.3.16 Time XMLHelper::getElementTimestampValue (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a **Time**(p. 516) representation of its value subelement.

Used to extract data from ValueType descendants.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The **Time**(p. 516) representation of the value element's content.

4.194.3.17 DOMElement * XMLHelper::getFirstChildByTag (const DOMElement & *n*, const std::string & *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The first subelement with a matching tag of null if no such element was found..

4.194.3.18 DOMElement * XMLHelper::getFirstChildByTag (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The first subelement with a matching tag of null if no such element was found..

4.194.3.19 int XMLHelper::getInt (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int representation its content.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:
The int representation of the subelement's content.

4.194.3.20 int XMLHelper::getIntAttribute (const DOMElement & *n*, const char * *tag*) [inline, static]

Gets an int representation of the attribute named 'tag' of the provided DOMElement.

Parameters:
n The DOMElement to get the attribute from.

tag The name of the attribute.

Returns:
The int representation of the attribute.

4.194.3.21 int64_t XMLHelper::getLongInt (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a int64_t representation its content.

Parameters:
n The parent DOMElement.

tag The tag of the subelement.

Returns:
The int64_t representation of the subelement's content.

4.194.3.22 Shape * XMLHelper::getShape (const DOMElement & *n*, const Reference & *scope*) [inline, static]

Gets a **Shape**(p. 412) representation of the provided DOMElement.

Parameters:
n The parent DOMElement.

scope The **Reference**(p. 378) to the scope this shape should live in.

Returns:
The **Shape**(p. 412) representation of the element's content.

4.194.3.23 Shape * XMLHelper::getShape (const DOMElement & *n*, const char * *tag*, const Reference & *scope*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a **Shape**(p. 412) representation its content.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

scope The **Reference**(p. 378) to the scope this shape should live in.

Returns:

The **Shape**(p. 412) representation of the subelement's content.

4.194.3.24 void XMLHelper::getString (const DOMElement & *n*, const char * *tag*, std::string & *outStr*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a string representation its content.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

outStr The string representation of the subelement's content.

4.194.3.25 string XMLHelper::getStringAttribute (const DOMElement & *n*, const char * *tag*) [inline, static]

Gets a string representation of the attribute named 'tag' of the provided DOMElement.

Parameters:

n The DOMElement to get the attribute from.

tag The name of the attribute.

Returns:

The string representation of the attribute.

4.194.3.26 Time XMLHelper::getTime (const DOMElement & *n*, const char * *tag*) [inline, static]

Finds the first subelement of the provided DOMElement that has a tag that matches the specified tag and returns a **Time**(p. 516) representation of its contents.

Used to extract data from ValueType descendants.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The **Time**(p. 516) representation of the subelement's content.

4.194.3.27 string XMLHelper::getTypeAttribute (const DOMElement & *n*) [inline, static]

Convenience function for getting the xsi:type attribute and stripping the leading namespace from it.

Parameters:

n The DOMElement to get the type attribute from.

Returns:

The string representation of type.

4.194.3.28 const XMLCh * XMLHelper::getXMLChString (const DOMElement & *n*, const char * *tag*) [inline, static]

Gets an XMLCh string from the first subelement of the provided DOMElement that has the tag 'tag'.

Parameters:

n The parent DOMElement.

tag The tag of the subelement.

Returns:

The XMLCh string or null if no element with the specified tag was found.

4.194.3.29 string XMLHelper::nodeTypeToString (int *i*) [inline, static]

Maps node types to their names.

Parameters:

i The node type.

Returns:

A string with the name of the node type.

4.194.3.30 string & XMLHelper::removeNamespace (std::string & *s*) [inline, static]

Removes the namespace from a string.

Works by removing all charachters up to and including the last ':' character.

Parameters:

s The string to remove the namespace from.

Returns:

The same string with the namespace removed.

4.194.3.31 void XMLHelper::timeToDateTime (std::ostream & *o*, Time *time*) [inline, static]

Prints a **Time**(p. 516) object to the provided stream in XML Schema dateTime format. Always assumes UTC.

Parameters:

o The stream to write to.

time The **Time**(p. 516) object to write.

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

- XMLHelper.h
- XMLHelper.cpp

4.195 XSDContent Class Reference

This class represents the contents of an xml schema document.

`#include <XSDContent.h>`

Public Member Functions

- **XSDContent** ()

  Default constructor.

- ~**XSDContent** ()

  Destructor.

- const **Type** & **getType** (const std::string &typeName)

  Creates a **Type**(p. 528) *object that corresponds to the xml schema type with the specified name.*

- const **Type** & **getType** (const std::string &typeName, const std::string &nameSpace)

  Creates a **Type**(p. 528) *object that corresponds to the xml schema type with the specified name and namespace.*

Static Public Member Functions

- **XSDContent** * **createFromFile** (const std::string &filename)

  Reads an xml schema from a file.

- **XSDContent** * **createFromString** (const std::string &schemastring)

  Reads an xml schema from a string.

- **XSDContent** * **create** (InputSource *schemaSource)

  Reads an xml schema from an InputSource.

Private Member Functions

- void **parseSchema** (const InputSource &source)

  Parses the schema.

Private Attributes

- XMLGrammarPool * **mGrammarPool**

  The grammar pool to use.

- XSModel * **mXSModel**

  The XSModel created from the xml schema.

- const std::string **mNamespace**

The namespace.

- std::map< std::string, **Type** * > **mResolvedTypes**
  Map(p. 292) *containing the types already resolved.*

4.195.1 Detailed Description

This class represents the contents of an xml schema document.

Author:
Per Alexius

Date **Date**
2006/07/21 13:35:30

4.195.2 Member Function Documentation

4.195.2.1 XSDContent * XSDContent::create (InputSource * *schemaSource*) [static]

Reads an xml schema from an InputSource.

Parameters:
schemaSource The input source containing the schema.

Returns:
The newly created XSDContent.

4.195.2.2 XSDContent * XSDContent::createFromFile (const std::string & *filename*) [static]

Reads an xml schema from a file.

Parameters:
filename The name of the file.

Returns:
The newly created XSDContent.

4.195.2.3 XSDContent * XSDContent::createFromString (const std::string & *schemastring*) [static]

Reads an xml schema from a string.

Parameters:
schemastring The string containing the schema.

Returns:
The newly created XSDContent.

4.195.2.4 const Type & XSDContent::getType (const std::string & *typeName*, const std::string & *nameSpace*)

Creates a **Type**(p. 528) object that corresponds to the xml schema type with the specified name and namespace.

Parameters:

typeName The name of the **Type**(p. 528).

nameSpace The namespace of the **Type**(p. 528).

Returns:

The **Type**(p. 528) with the specified name and namespace.

4.195.2.5 const Type & XSDContent::getType (const std::string & *typeName*)

Creates a **Type**(p. 528) object that corresponds to the xml schema type with the specified name.

Parameters:

typeName The name of the **Type**(p. 528).

Returns:

The **Type**(p. 528) with the specified name.

4.195.2.6 void XSDContent::parseSchema (const InputSource & *source*) [private]

Parses the schema.

Parameters:

source The input source to create the XSDContent from.

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

- XSDContent.h
- XSDContent.cpp

4.196 XStr Class Reference

This is a simple class for transcoding of char arrays to XMLCh strings.

`#include <StrX.h>`

Public Member Functions

- **XStr** (const char *const toTranscode)

  Constructor that transcodes a char array to an XMLCh string.

- **XStr** (const std::string toTranscode)

  Constructor that transcodes a std::string to an XMLCh string.

- ~**XStr** ()

  Destructor.

- const XMLCh * **str** () const

  Accessor for the string.

Private Member Functions

- void **transcode** (const char *const toTranscode)

  Converts the provided string to XMLCh using the XMLTranscoder given by **TranscoderWrapper::getTranscoder()**(p. 525).

Private Attributes

- XMLCh **mStr** [**kBlock**]

  Buffer(p. 67) *for the string in XMLCh array form.*

- XMLCh * **mLongStr**

  Pointer to allocated memory if string is to large to fit in mStr.

- unsigned char **mNumBytesPerChar** [**kBlock**]

  Never really used but required by transcodeFrom() call.

Static Private Attributes

- const unsigned int **kBlock** = 256

  Default size of a block to transcode.

4.196.1 Detailed Description

This is a simple class for transcoding of char arrays to XMLCh strings.

Inspired (very much) by the **StrX**(p. 498) class in the DOMCount example in the Xerces-c distribution

Author:
Per Alexius

Date:
2006/09/12 11:54:20

4.196.2 Constructor & Destructor Documentation

4.196.2.1 XStr::XStr (const char *const *toTranscode*) [inline]

Constructor that transcodes a char array to an XMLCh string.

Parameters:
toTranscode The string to transcode

4.196.2.2 XStr::XStr (const std::string *toTranscode*) [inline]

Constructor that transcodes a std::string to an XMLCh string.

Parameters:
toTranscode The string to transcode

4.196.3 Member Function Documentation

4.196.3.1 const XMLCh* XStr::str () const [inline]

Accessor for the string.

Returns:
The string as an XMLCh array.

4.196.3.2 void XStr::transcode (const char *const *toTranscode*) [inline, private]

Converts the provided string to XMLCh using the XMLTranscoder given by **TranscoderWrapper::getTranscoder()**(p. 525).

Parameters:
toTranscode The string to transcode.

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

- StrX.h

Chapter 5

Stratmas File Documentation

5.1 GoodStuff.h File Reference

This file contains some useful constants and functions.

`#include <cmath>`

`#include <algorithm>`

Defines

- #define **ByteSwap**(x) ByteSwapX((unsigned char *) &x,sizeof(x))

  For swapping endian of data type of any size.

- #define **isnan**(x) (isnan(x))

Functions

- int **Round** (float x)

  Poor round function adapted from older versions of Stratmas.

- int **Round** (double x)

  Poor round function adapted from older versions of Stratmas.

- int **between** (int x, int bot, int top)

  Makes sure x fits in the intervall [bot, top]. If it doesn't -round x to either of bot or top that is closest.

- double **between** (double x, double bot, double top)

  Makes sure x fits in the intervall [bot, top]. If it doesn't -round x to either of bot or top that is closest.

- void **ByteSwapX** (unsigned char *b, int n)

  For swapping byte order of data type of any size.

Variables

- const double **kPi** = (4.0 * std::atan2(1.0, 1.0))
  pi
- const double **k2Pi** = 2 * **kPi**
  *2 * pi*
- const double **kDeg2Rad** = **kPi** / 180.0
  For converting degrees to radians.
- const double **kRad2Deg** = 180.0 / **kPi**
  For converting radians to degrees.
- const double **kKmPerDegreeLat** = 40008.0 / 360.0
  Approximate number of kilometers per degree latitude.
- const double **kMetersPerDegreeLat** = 40008000.0 / 360.0
  Approximate number of meters per degree latitude.
- const double **kDegreesLatPerMeter** = 1.0 / **kMetersPerDegreeLat**
  Approximate number of degrees latitude per meter.
- const double **kKmPerDegreeLat2** = **kKmPerDegreeLat** * **kKmPerDegreeLat**
  Square of approximate number of kilometers per degree latitude.
- const double **kMetersPerDegreeLat2** = **kMetersPerDegreeLat** * **kMetersPerDegreeLat**
  Square of approximate number of meters per degree latitude.

5.1.1 Detailed Description

This file contains some useful constants and functions.

5.1.2 Function Documentation

5.1.2.1 double between (double *x*, double *bot*, double *top*) [inline]

Makes sure x fits in the intervall [bot, top]. If it doesn't -round x to either of bot or top that is closest.

Parameters:

x The value to be checked

bot The lower bound

top The upper bound

Returns:

A double value in the intervall [bot, top]

5.1.2.2 int between (int *x*, int *bot*, int *top*) [inline]

Makes sure x fits in the intervall [bot, top]. If it doesn't -round x to either of bot or top that is closest.

Parameters:

x The value to be checked

bot The lower bound

top The upper bound

Returns:

An integer value in the intervall [bot, top]

5.1.2.3 void ByteSwapX (unsigned char * *b*, int *n*) [inline]

For swapping byte order of data type of any size.

Parameters:

b The data to swap byte order for.

n The number of bytes of data.

5.1.2.4 int Round (double *x*) [inline]

Poor round function adapted from older versions of Stratmas.

Parameters:

x The value to be rounded.

Returns:

The rounded value.

5.1.2.5 int Round (float *x*) [inline]

Poor round function adapted from older versions of Stratmas.

Parameters:

x The value to be rounded.

Returns:

The rounded value.

5.2 random.h File Reference

This file contains some useful functions for handling random numbers and different probability distributions.

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include "LogStream.h"
```

Classes

- class **GaussSaver**

  Helper class for storing info of the number that is saved by the gaussian random number algorithm.

- class **PrivateRandom**

  Helper class for handling random numbers that should not interfere with the sequence of random numbers generated during a simlation.

Functions

- void **setRandomSeed** (unsigned long seed)

  Sets the random seed.

- unsigned long **createRandomSeed** ()

  Creates a seed for srandom based on time().

- double **RandomUniform** ()

  Returns a random double that is uniformly distributed between zero and one.

- double **RandomUniform** (double low, double high)

  Returns a random double that is uniformly distributed between specified limits.

- double **Gaussian** (double scale, double mean=0.0)

  Return a random double with a normal density centered on the mean, with standard deviation = scale.

- int **Poisson** (double lambda)

  Return a random integer with a Poisson distribution.

- double **XPoisson** (double lambda, double clip)

  Return a random integer with a Poisson distribution (truncated).

- double **Exponential** (double mean)

  Return a random integer with an Exponential distribution.

- int **probBySize** (const double *sizes, int length)

  Chooses one element in an array with a probability that is proportional to the size of that element.

Variables

- const double **kRandomMax** = static_cast<double>(RAND_MAX)

  The maximum value that may be returned by random().

5.2.1 Detailed Description

This file contains some useful functions for handling random numbers and different probability distributions.

"

Author:
Per Alexius

Date **Date**
2006/09/05 14:18:21

5.2.2 Function Documentation

5.2.2.1 double Exponential (double *mean*) [inline]

Return a random integer with an Exponential distribution.

Author:
Loren Cobb

Parameters:
mean The mean value

Returns:
A random double with an Exponential distribution.

5.2.2.2 double Gaussian (double *scale*, double *mean*) [inline]

Return a random double with a normal density centered on the mean, with standard deviation = scale.

Author:
Loren Cobb

Parameters:
scale Standard deviation

mean Mean

Returns:
A random double with a normal density.

5.2.2.3 int Poisson (double *lambda*) [inline]

Return a random integer with a Poisson distribution.

Author:
Loren Cobb

Parameters:
lambda The expected value of the distribution.

Returns:
A random double with a Poisson distribution.

5.2.2.4 int probBySize (const double * *sizes*, int *length*) [inline]

Chooses one element in an array with a probability that is proportional to the size of that element.

Parameters:
sizes An array of doubles representing the sizes.

length The number of elements in the array.

Returns:
The index of the element that was choosen.

5.2.2.5 double RandomUniform (double *low*, double *high*) [inline]

Returns a random double that is uniformly distributed between specified limits.

Author:
Loren Cobb

Parameters:
low Lower bound

high Upper bound

Returns:
A random double that is uniformly distributed in the intervall [low, high]

5.2.2.6 double RandomUniform () [inline]

Returns a random double that is uniformly distributed between zero and one.

Returns:
A random double that is uniformly distributed between zero and one.

5.2.2.7 void setRandomSeed (unsigned long *seed*) [inline]

Sets the random seed.

Parameters:
seed The new random seed.

5.2.2.8 double XPoisson (double *lambda*, double *clip*) [inline]

Return a random integer with a Poisson distribution (truncated).

Author:
Loren Cobb

Parameters:
lambda The expected value of the distribution.

clip The point of truncation.

Returns:
A random double with a Poisson distribution (truncated).

5.3 stratmas.cpp File Reference

This file contains the main function of the StratmasServer.

```
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include "Registrator.h"
#include "debugheader.h"
#include "Environment.h"
#include "IOHandler.h"
#include "LogStream.h"
#include "PropertyHandler.h"
#include "PVInfo.h"
#include "random.h"
#include "Server.h"
```

Functions

- void **initTimeZone** ()
- int **main** (int argc, char **argv)

  The main function.

5.3.1 Detailed Description

This file contains the main function of the StratmasServer.

5.3.2 Function Documentation

5.3.2.1 int main (int *argc*, char ** *argv*)

The main function.

Parameters:

argc argc

argv argv

Returns:

Exit status.

5.4 StratmasConstants.h File Reference

This file contains some constants that are used by the Stratmas simulation.

Variables

- const double **kMinPopulation** = 0.5

  Minimum population a cell must have for the simulation to update it.

- const double **kMaxStoredFood** = 15.0

  Maximum days of locally stored food.

- const double **kMaxStoredWater** = 5.0

  Maximum days of locally stored clean water.

- const double **kNoCampZone** = 100000.0

  Do not build a new camp within this distance (m) of any other.

- const double **kFractionProtectedResettling** = 0.005

  This is the fraction of the protected people in each cell that resettles every day. The resettling destination is based on the initial population of each cell.

- const double **kDiffusionThreatThreshold** = 65.0

  If the threat level rises above hcDiffusionThreatThreshold, people start moving to neighboring cells with lower threat.

- const double **kDiffusionFractionUnshelteredMoving** = 0.20

  This is the fraction of the unsheltered people that moves when the threat level rises above kDiffusionThreatThreshold moves.

- const double **kDiffusionFractionAtHomeMoving** = 0.10

  This is the fraction of the people still living at home that moves when the threat level rises above kDiffusionThreatThreshold moves.

- const double **kRefugeeMeanSpeed** = 30000.0

  Mean speed of refugee movements in m/day.

- const double **kRefugeeSpeedStandardDeviation** = 15000.0

  Standard deviation of refugee movement speed.

- const double **kFractionMovingRefugees** = 0.40

  This is the fraction of refugees that moves towards a camp.

- const double **kFoodPPPDKg** = 1.0

  Food consumed per person per day in kg.

5.4.1 Detailed Description

This file contains some constants that are used by the Stratmas simulation.

Index