

d3sf

Generated by Doxygen 1.6.3

Sat Apr 17 14:24:14 2010

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	node::in_queues Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Member Data Documentation	5
3.1.2.1	insert	5
3.1.2.2	message	5
3.1.2.3	remove	5
3.2	node Class Reference	6
3.2.1	Detailed Description	8
3.2.2	Constructor & Destructor Documentation	8
3.2.2.1	node	8
3.2.3	Member Function Documentation	8
3.2.3.1	broadcast_neigh	8
3.2.3.2	get_neighbours	8
3.2.3.3	get_number_inserts	8
3.2.3.4	get_number_messages	9
3.2.3.5	get_number_removes	9
3.2.3.6	insert	9
3.2.3.7	is_neighbour	9
3.2.3.8	logic	9
3.2.3.9	logic_init	9
3.2.3.10	message	10
3.2.3.11	on_insert	10

3.2.3.12	on_message_receive	10
3.2.3.13	on_remove	10
3.2.3.14	process	10
3.2.3.15	process_inserts	11
3.2.3.16	process_messages	11
3.2.3.17	process_removes	11
3.2.3.18	remove	11
3.2.3.19	send_message	11
3.2.3.20	tick	11
3.2.4	Member Data Documentation	11
3.2.4.1	action_buffer	11
3.2.4.2	logical_time	12
3.2.4.3	neighbours	12
3.3	world::node_info Struct Reference	13
3.3.1	Detailed Description	13
3.3.2	Member Data Documentation	13
3.3.2.1	id	13
3.3.2.2	obj	13
3.4	node::queue_edge Struct Reference	14
3.4.1	Detailed Description	14
3.4.2	Member Data Documentation	14
3.4.2.1	from	14
3.4.2.2	time	14
3.4.2.3	to	14
3.5	node::queue_message Struct Reference	15
3.5.1	Detailed Description	15
3.5.2	Member Data Documentation	15
3.5.2.1	from	15
3.5.2.2	message	15
3.5.2.3	time	15
3.6	node::queue_remove Struct Reference	16
3.6.1	Detailed Description	16
3.6.2	Member Data Documentation	16
3.6.2.1	time	16
3.6.2.2	to	16
3.7	world Class Reference	17

3.7.1	Detailed Description	18
3.7.2	Constructor & Destructor Documentation	18
3.7.2.1	world	18
3.7.2.2	world	18
3.7.3	Member Function Documentation	18
3.7.3.1	converge	18
3.7.3.2	get_number_inserts	18
3.7.3.3	get_number_messages	19
3.7.3.4	get_number_removes	19
3.7.3.5	insert_edge	19
3.7.3.6	insert_node	19
3.7.3.7	lookup_id	19
3.7.3.8	print_list	19
3.7.3.9	print_nodes	20
3.7.3.10	tick	20
3.7.4	Member Data Documentation	20
3.7.4.1	id_counter	20
3.7.4.2	nodes	20
3.7.4.3	tick_counter	20
4	File Documentation	21
4.1	debug.hpp File Reference	21
4.1.1	Define Documentation	21
4.1.1.1	dbg	21
4.1.1.2	dbg_var	21
4.2	node.cpp File Reference	22
4.2.1	Define Documentation	22
4.2.1.1	FILENAME	22
4.3	node.hpp File Reference	23
4.4	random_graph.cpp File Reference	24
4.4.1	Function Documentation	24
4.4.1.1	main	24
4.5	sample.cpp File Reference	25
4.5.1	Function Documentation	25
4.5.1.1	main	25
4.6	world.cpp File Reference	26
4.6.1	Define Documentation	26

4.6.1.1	FILENAME	26
4.6.2	Variable Documentation	26
4.6.2.1	changes	26
4.7	world.hpp File Reference	27

Chapter 1

Class Index

1.1 Class List

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

node::in_queues (Incoming action queues)	5
node (Class for all nodes)	6
world::node_info (Information about the node)	13
node::queue_edge (Structure to save incoming "insert" calls)	14
node::queue_message (Structure to save incoming "message" calls)	15
node::queue_remove (Structure to save incoming "remove" calls)	16
world (Consists of all nodes and some extra information)	17

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

debug.hpp	21
node.cpp	22
node.hpp	23
random_graph.cpp	24
sample.cpp	25
world.cpp	26
world.hpp	27

Chapter 3

Class Documentation

3.1 `node::in_queues` Struct Reference

incoming action queues

Public Attributes

- `std::queue< struct queue_edge >` `insert`
- `std::queue< struct queue_remove >` `remove`
- `std::queue< struct queue_message >` `message`

3.1.1 Detailed Description

incoming action queues In this structure all the incoming calls are stored until the next round starts

3.1.2 Member Data Documentation

3.1.2.1 `std::queue<struct queue_edge>` `node::in_queues::insert`

3.1.2.2 `std::queue<struct queue_message>` `node::in_queues::message`

3.1.2.3 `std::queue<struct queue_remove>` `node::in_queues::remove`

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

- [node.hpp](#)

3.2 node Class Reference

class for all nodes

```
#include <node.hpp>
```

Classes

- struct [in_queues](#)
incoming action queues
- struct [queue_edge](#)
structure to save incoming "insert" calls
- struct [queue_message](#)
structure to save incoming "message" calls
- struct [queue_remove](#)
structure to save incoming "remove" calls

Public Member Functions

- [node](#) ()
Constructor.
- `std::list< node * > * get_neighbours ()`
Getter for neighbours.
- `void insert (node *n_a, node *n_b)`
inserts an edge
- `void remove (node *n_a)`
removes edge
- `void message (node *origin, const char *msg)`
Transmit a message.
- `void tick ()`
tick increases the logical clock
- `int process ()`
processes all incoming queues and calls [logic\(\)](#)
- `unsigned long get_number_messages ()`
get the number of transmitted messages
- `unsigned long get_number_removes ()`
get the number of removed edges

- unsigned long `get_number_inserts ()`
get the number of inserted edges

Private Member Functions

- int `process_inserts ()`
process all inserts from the last round
- int `process_removes ()`
process all removes from the last round
- int `process_messages ()`
process all messages from the last round
- void `send_message (node *destination, unsigned int type, const char *message)`
sends a message
- int `is_neighbour (node *search)`
checks if the node is a neighbour
- void `broadcast_neigh (unsigned int message_type, const char *message, const node *exclude)`
broadcast to neighbours
- void `logic_init ()`
initiate logic
- void `on_message_receive (node *origin, unsigned int type, char *msg)`
message receive event
- void `on_insert (node *new_neighbour)`
insert event
- void `on_remove (node *to_remove)`
remove event
- int `logic ()`
Executed once a round.

Private Attributes

- unsigned int `logical_time`
logical time
- `std::list< node * > neighbours`
neighbours of this node stored in a list of pointers

- struct `node::in_queues action_buffer`
incoming action queues

3.2.1 Detailed Description

class for all nodes This class contains all structural and logical elements of the distributed nodes in the graph.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 `node::node ()`

Constructor.

3.2.3 Member Function Documentation

3.2.3.1 `void node::broadcast_neigh (unsigned int message_type, const char * message, const node * exclude) [private]`

broadcast to neighbours

Parameters

- message_type* type of the message (4 byte)
- message* the message itself as string
- exclude* excludes this node from the broadcast

3.2.3.2 `std::list< node * > * node::get_neighbours ()`

Getter for neighbours.

Don't use this for node logic. This would not model a proper distributed system.

Returns

A list of node pointers is returned.

3.2.3.3 `unsigned long node::get_number_inserts ()`

get the number of inserted edges

Returns

number of inserted edges

3.2.3.4 unsigned long node::get_number_messages ()

get the number of transmitted messages

Returns

number of transmitted messages

3.2.3.5 unsigned long node::get_number_removes ()

get the number of removed edges

Returns

number of removed edges

3.2.3.6 void node::insert (node * *n_a*, node * *n_b*)

inserts an edge

This function inserts an edge from *n_a* to *n_b* if this object knows *n_a*.

Parameters

n_a The origin of the edge

n_b The destination of the edge

3.2.3.7 int node::is_neighbour (node * *search*) [private]

checks if the node is a neighbour

Parameters

search the node to search for

3.2.3.8 int node::logic () [private]

Executed once a round.

In this function you can place your node logic. The function is called every round.

Returns

1 if something changed else 0

3.2.3.9 void node::logic_init () [private]

initiate logic

3.2.3.10 void node::message (node * *origin*, const char * *msg*)

Transmit a message.

For sending messages you should use `send_message`.

Parameters

origin Origin of the message, this should always be "this"

msg This is the message including the header (type) of the message

3.2.3.11 void node::on_insert (node * *new_neighbour*) [private]

insert event

This function is called after the new edge to the node is inserted

Parameters

new_neighbour The destination of the new edge

3.2.3.12 void node::on_message_receive (node * *origin*, unsigned int *type*, char * *msg*) [private]

message receive event

this function is called if a message is received. This function is one of two logic functions you can fill with logic for the behaviour of your nodes.

Parameters

origin Origin of the message (one of the neighbours).

type This is the 4 byte long message-type.

msg The content of the message as a string. So be carefull, you can't use '0' in the message.

3.2.3.13 void node::on_remove (node * *to_remove*) [private]

remove event

This function is called before removing an element

Parameters

to_remove the node that will be removed from the neighbours list

3.2.3.14 int node::process ()

processes all incoming queues and calls `logic()`

Returns

Should return 0 if there are no outstanding tasks or actions done

3.2.3.15 int node::process_inserts () [private]

process all inserts from the last round

3.2.3.16 int node::process_messages () [private]

process all messages from the last round

3.2.3.17 int node::process_removes () [private]

process all removes from the last round

3.2.3.18 void node::remove (node * *n_a*)

removes edge

Removes the edge from this node to *n_a* if it exists

Parameters

n_a The destination node

3.2.3.19 void node::send_message (node * *destination*, unsigned int *type*, const char * *message*) [private]

sends a message

This function sends a message to the destination and combines the type and the message to the complete message.

Parameters

destination Destination of the message

type Type of the message

message The message itself

3.2.3.20 void node::tick ()

tick increases the logical clock

3.2.4 Member Data Documentation**3.2.4.1 struct node::in_queues node::action_buffer [private]**

incoming action queues

In this structure all the incoming calls are stored until the next round starts

3.2.4.2 `unsigned int node::logical_time` [`private`]

logical time

Every node has an autonomous logical time variable. They are different but this helps to mark the incoming actions with an relative time, so we don't process actions too early

3.2.4.3 `std::list<node*> node::neighbours` [`private`]

neighbours of this node stored in a list of pointers

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

- [node.hpp](#)
- [node.cpp](#)

3.3 world::node_info Struct Reference

information about the node

Public Attributes

- unsigned int [id](#)
unique id
- [node](#) * [obj](#)

3.3.1 Detailed Description

information about the node

3.3.2 Member Data Documentation

3.3.2.1 unsigned int world::node_info::id

unique id

this id is not available for the nodes themselves because of a real distributed system model.

3.3.2.2 node* world::node_info::obj

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

- [world.hpp](#)

3.4 node::queue_edge Struct Reference

structure to save incoming "insert" calls

Public Attributes

- [node * from](#)
- [node * to](#)
- unsigned int [time](#)
logical receive time

3.4.1 Detailed Description

structure to save incoming "insert" calls

3.4.2 Member Data Documentation

3.4.2.1 node* node::queue_edge::from

3.4.2.2 unsigned int node::queue_edge::time

logical receive time

3.4.2.3 node* node::queue_edge::to

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

- [node.hpp](#)

3.5 node::queue_message Struct Reference

structure to save incoming "message" calls

Public Attributes

- [node * from](#)
- [char * message](#)
the message to process
- [unsigned int time](#)
logical receive time

3.5.1 Detailed Description

structure to save incoming "message" calls

3.5.2 Member Data Documentation

3.5.2.1 node* node::queue_message::from

3.5.2.2 char* node::queue_message::message

the message to process

This message consists of a 4 byte header for the message-type.

3.5.2.3 unsigned int node::queue_message::time

logical receive time

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

- [node.hpp](#)

3.6 `node::queue_remove` Struct Reference

structure to save incoming "remove" calls

Public Attributes

- [node * to](#)
- unsigned int [time](#)
logical receive time

3.6.1 Detailed Description

structure to save incoming "remove" calls

3.6.2 Member Data Documentation

3.6.2.1 unsigned int `node::queue_remove::time`

logical receive time

3.6.2.2 `node* node::queue_remove::to`

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

- [node.hpp](#)

3.7 world Class Reference

consists of all nodes and some extra information

```
#include <world.hpp>
```

Classes

- struct `node_info`
information about the node

Public Member Functions

- `world ()`
Constructor.
- `world (const char *path)`
Constructor.
- void `insert_edge (unsigned int a, unsigned int b)`
inserts an edge
- `node * insert_node (unsigned int id)`
inserts the given node if it doesn't exist
- void `tick ()`
triggers a "round"
- void `print_nodes ()`
Prints all id's of nodes.
- void `print_list ()`
Prints an adjacency list that is in the correct format for the world-constructor.
- unsigned int `lookup_id (node *tgt)`
Looks up the id from the pointer.
- unsigned int `converge (unsigned int max_ticks)`
let the algorithm converge
- unsigned long `get_number_messages ()`
get the number of transmitted messages
- unsigned long `get_number_removes ()`
get the number of removed edges
- unsigned long `get_number_inserts ()`
get the number of inserted edges

Private Attributes

- `std::list< struct node_info > nodes`
- `unsigned int id_counter`
the maximum of all managed id's
- `unsigned int tick_counter`

3.7.1 Detailed Description

consists of all nodes and some extra information

3.7.2 Constructor & Destructor Documentation

3.7.2.1 `world::world ()`

Constructor.

3.7.2.2 `world::world (const char * path)`

Constructor.

Builds a directed graph from the given adjacency list file.

Parameters

path Path to the adjacency list file

3.7.3 Member Function Documentation

3.7.3.1 `unsigned int world::converge (unsigned int max_ticks)`

let the algorithm converge

Parameters

max_ticks maximal executed ticks

Returns

ticks needed to converge to a stable graph (a graph without any actions anymore.

3.7.3.2 `unsigned long world::get_number_inserts ()`

get the number of inserted edges

Returns

number of inserted edges

3.7.3.3 unsigned long world::get_number_messages ()

get the number of transmitted messages

Returns

number of transmitted messages

3.7.3.4 unsigned long world::get_number_removes ()

get the number of removed edges

Returns

number of removed edges

3.7.3.5 void world::insert_edge (unsigned int *a*, unsigned int *b*)

inserts an edge

inserts an edge from id *a* to id *b* and adds the nodes if they don't exist

Parameters

a Origin of the edge

b destination of the edge

3.7.3.6 node * world::insert_node (unsigned int *id*)

inserts the given node if it doesn't exist

Parameters

id Uses this as id for the node

Returns

The constructed or already existing node

3.7.3.7 unsigned int world::lookup_id (node * *tgt*)

Looks up the id from the pointer.

Parameters

tgt The pointer to the node that should be looked up

3.7.3.8 void world::print_list ()

Prints an adjacency list that is in the correct format for the world-constructor.

Some small note: This function is not the fastest because it makes lookups of the id.

3.7.3.9 void world::print_nodes ()

Prints all id's of nodes.

3.7.3.10 void world::tick ()

triggers a "round"

This function simulates an round in the graph. This means on all managed nodes the function [tick\(\)](#) is called in the first step. Then all process() functions are called.

3.7.4 Member Data Documentation

3.7.4.1 unsigned int world::id_counter [private]

the maximum of all managed id's

3.7.4.2 std::list<struct node_info> world::nodes [private]

3.7.4.3 unsigned int world::tick_counter [private]

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

- [world.hpp](#)
- [world.cpp](#)

Chapter 4

File Documentation

4.1 debug.hpp File Reference

Defines

- #define `dbg`(section, format) ;
- #define `dbg_var`(section, type, var) ;

4.1.1 Define Documentation

4.1.1.1 #define `dbg`(section, format) ;

4.1.1.2 #define `dbg_var`(section, type, var) ;

4.2 node.cpp File Reference

```
#include <cstdlib>
#include <cstring>
#include <stdio.h>
#include "node.hpp"
#include <list>
#include <queue>
```

Defines

- #define [FILENAME](#) "node.cpp"

4.2.1 Define Documentation

4.2.1.1 #define FILENAME "node.cpp"

4.3 node.hpp File Reference

```
#include <list>
#include <queue>
```

Classes

- class [node](#)
class for all nodes
- struct [node::queue_edge](#)
structure to save incoming "insert" calls
- struct [node::queue_message](#)
structure to save incoming "message" calls
- struct [node::queue_remove](#)
structure to save incoming "remove" calls
- struct [node::in_queues](#)
incoming action queues

4.4 random_graph.cpp File Reference

```
#include "node.hpp"  
#include "world.hpp"  
#include <list>  
#include "debug.hpp"  
#include <stdlib.h>  
#include <time.h>  
#include <stdio.h>
```

Functions

- int [main](#) (int *cargs*, char ***argv*)

4.4.1 Function Documentation

4.4.1.1 int main (int *cargs*, char ** *argv*)

4.5 sample.cpp File Reference

```
#include "node.hpp"  
#include "world.hpp"  
#include "debug.hpp"  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>
```

Functions

- `int main (int cargs, char **argv)`

4.5.1 Function Documentation

4.5.1.1 `int main (int cargs, char ** argv)`

4.6 world.cpp File Reference

```
#include <list>
#include <cstdlib>
#include <stdio.h>
#include "node.hpp"
#include "world.hpp"
#include "debug.hpp"
#include <unistd.h>
```

Defines

- #define [FILENAME](#) "world.cpp"

Variables

- int [changes](#)

4.6.1 Define Documentation

4.6.1.1 #define FILENAME "world.cpp"

4.6.2 Variable Documentation

4.6.2.1 int changes

4.7 world.hpp File Reference

```
#include <list>
#include "node.hpp"
```

Classes

- class [world](#)
consists of all nodes and some extra information
- struct [world::node_info](#)
information about the node

Index

- action_buffer
 - node, 11
- broadcast_neigh
 - node, 8
- changes
 - world.cpp, 26
- converge
 - world, 18
- dbg
 - debug.hpp, 21
- dbg_var
 - debug.hpp, 21
- debug.hpp, 21
 - dbg, 21
 - dbg_var, 21
- FILENAME
 - node.cpp, 22
 - world.cpp, 26
- from
 - node::queue_edge, 14
 - node::queue_message, 15
- get_neighbours
 - node, 8
- get_number_inserts
 - node, 8
 - world, 18
- get_number_messages
 - node, 8
 - world, 18
- get_number_removes
 - node, 9
 - world, 19
- id
 - world::node_info, 13
- id_counter
 - world, 20
- insert
 - node, 9
 - node::in_queues, 5
- insert_edge
 - world, 19
- insert_node
 - world, 19
- is_neighbour
 - node, 9
- logic
 - node, 9
- logic_init
 - node, 9
- logical_time
 - node, 11
- lookup_id
 - world, 19
- main
 - random_graph.cpp, 24
 - sample.cpp, 25
- message
 - node, 9
 - node::in_queues, 5
 - node::queue_message, 15
- neighbours
 - node, 12
- node, 6
 - action_buffer, 11
 - broadcast_neigh, 8
 - get_neighbours, 8
 - get_number_inserts, 8
 - get_number_messages, 8
 - get_number_removes, 9
 - insert, 9
 - is_neighbour, 9
 - logic, 9
 - logic_init, 9
 - logical_time, 11
 - message, 9
 - neighbours, 12
 - node, 8
 - on_insert, 10
 - on_message_receive, 10
 - on_remove, 10
 - process, 10
 - process_inserts, 10

- process_messages, 11
- process_removes, 11
- remove, 11
- send_message, 11
- tick, 11
- node.cpp, 22
 - FILENAME, 22
- node.hpp, 23
- node::in_queues, 5
 - insert, 5
 - message, 5
 - remove, 5
- node::queue_edge, 14
 - from, 14
 - time, 14
 - to, 14
- node::queue_message, 15
 - from, 15
 - message, 15
 - time, 15
- node::queue_remove, 16
 - time, 16
 - to, 16
- nodes
 - world, 20
- obj
 - world::node_info, 13
- on_insert
 - node, 10
- on_message_receive
 - node, 10
- on_remove
 - node, 10
- print_list
 - world, 19
- print_nodes
 - world, 19
- process
 - node, 10
- process_inserts
 - node, 10
- process_messages
 - node, 11
- process_removes
 - node, 11
- random_graph.cpp, 24
 - main, 24
- remove
 - node, 11
 - node::in_queues, 5
- sample.cpp, 25
 - main, 25
- send_message
 - node, 11
- tick
 - node, 11
 - world, 20
- tick_counter
 - world, 20
- time
 - node::queue_edge, 14
 - node::queue_message, 15
 - node::queue_remove, 16
- to
 - node::queue_edge, 14
 - node::queue_remove, 16
- world, 17
 - converge, 18
 - get_number_inserts, 18
 - get_number_messages, 18
 - get_number_removes, 19
 - id_counter, 20
 - insert_edge, 19
 - insert_node, 19
 - lookup_id, 19
 - nodes, 20
 - print_list, 19
 - print_nodes, 19
 - tick, 20
 - tick_counter, 20
 - world, 18
- world.cpp, 26
 - changes, 26
 - FILENAME, 26
- world.hpp, 27
- world::node_info, 13
 - id, 13
 - obj, 13