

zergmap

Generated by Doxygen 1.8.11

Contents

1	README	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	_item Struct Reference	7
4.1.1	Field Documentation	7
4.1.1.1	data	7
4.1.1.2	priority	7
4.2	_map Struct Reference	7
4.2.1	Field Documentation	8
4.2.1.1	capacity	8
4.2.1.2	data	8
4.2.1.3	size	8
4.3	_pqueue Struct Reference	8
4.3.1	Field Documentation	9
4.3.1.1	capacity	9
4.3.1.2	cmp	9
4.3.1.3	data	9
4.3.1.4	size	9
4.4	_vmap Struct Reference	9

4.4.1	Field Documentation	9
4.4.1.1	capacity	9
4.4.1.2	data	9
4.4.1.3	size	9
4.5	gpsPayload::accuracy Union Reference	10
4.5.1	Field Documentation	10
4.5.1.1	fAccuracy	10
4.5.1.2	iAccuracy	10
4.6	gpsPayload::altitude Union Reference	10
4.6.1	Field Documentation	10
4.6.1.1	fAltitude	10
4.6.1.2	iAltitude	10
4.7	gpsPayload::bearing Union Reference	10
4.7.1	Field Documentation	11
4.7.1.1	fBearing	11
4.7.1.2	iBearing	11
4.8	cPayload Struct Reference	11
4.8.1	Field Documentation	12
4.8.1.1	command	12
4.8.1.2	param1	12
4.8.1.3	param2	12
4.9	edge_ Struct Reference	12
4.9.1	Field Documentation	12
4.9.1.1	next	12
4.9.1.2	to	12
4.9.1.3	weight	12
4.10	entry Struct Reference	13
4.10.1	Field Documentation	13
4.10.1.1	key	13
4.10.1.2	next	13

4.10.1.3	value	13
4.10.1.4	value	13
4.11	ethernetHeader Struct Reference	13
4.11.1	Field Documentation	14
4.11.1.1	destMac	14
4.11.1.2	etherType	14
4.11.1.3	sourceMac	14
4.12	gpsPayload Struct Reference	14
4.12.1	Field Documentation	15
4.12.1.1	accuracy	15
4.12.1.2	altitude	15
4.12.1.3	bearing	15
4.12.1.4	latitude	15
4.12.1.5	longitude	15
4.12.1.6	speed	15
4.13	graph Struct Reference	15
4.13.1	Field Documentation	16
4.13.1.1	root	16
4.14	ipv4Header Struct Reference	16
4.14.1	Field Documentation	16
4.14.1.1	checksum	16
4.14.1.2	destIp	16
4.14.1.3	dscp	16
4.14.1.4	flags	16
4.14.1.5	id	16
4.14.1.6	ipHeaderLength	16
4.14.1.7	ipLength	17
4.14.1.8	protocol	17
4.14.1.9	sourceIp	17
4.14.1.10	ttl	17

4.14.1.11 version	17
4.15 ipv6Header Struct Reference	17
4.15.1 Field Documentation	17
4.15.1.1 destination	17
4.15.1.2 flowLabel	17
4.15.1.3 hopLimit	17
4.15.1.4 nextHeader	17
4.15.1.5 payloadLength	17
4.15.1.6 source	17
4.15.1.7 trafficClass	17
4.15.1.8 version	17
4.16 gpsPayload::latitude Union Reference	18
4.16.1 Field Documentation	18
4.16.1.1 dLat	18
4.16.1.2 iLat	18
4.17 gpsPayload::longitude Union Reference	18
4.17.1 Field Documentation	18
4.17.1.1 dLong	18
4.17.1.2 iLong	18
4.18 node_ Struct Reference	19
4.18.1 Field Documentation	19
4.18.1.1 edges	19
4.18.1.2 name	19
4.18.1.3 next	19
4.19 cPayload::param2 Union Reference	19
4.19.1 Field Documentation	20
4.19.1.1 fParam2	20
4.19.1.2 iParam2	20
4.19.1.3 uiParam2	20
4.20 payload Struct Reference	20

4.20.1	Field Documentation	21
4.20.1.1	armor	21
4.20.1.2	currHitPoints	21
4.20.1.3	maxHitPoints	21
4.20.1.4	sSpeed	21
4.20.1.5	type	21
4.21	pcapFileHeader Struct Reference	21
4.21.1	Field Documentation	21
4.21.1.1	accDelta	21
4.21.1.2	fileTypeeld	21
4.21.1.3	gmtOffset	21
4.21.1.4	linkLayerType	21
4.21.1.5	majorVersion	21
4.21.1.6	maxLength	21
4.21.1.7	minorVersion	21
4.22	pcapPacketHeader Struct Reference	22
4.22.1	Field Documentation	22
4.22.1.1	fullLength	22
4.22.1.2	lengthOfData	22
4.22.1.3	microEpoch	22
4.22.1.4	unixEpoch	22
4.23	gpsPayload::speed Union Reference	22
4.23.1	Field Documentation	22
4.23.1.1	fSpeed	22
4.23.1.2	iSpeed	22
4.24	payload::sSpeed Union Reference	23
4.24.1	Field Documentation	23
4.24.1.1	fSpeed	23
4.24.1.2	iSpeed	23
4.25	udpHeader Struct Reference	23

4.25.1	Field Documentation	23
4.25.1.1	checksum	23
4.25.1.2	destPort	23
4.25.1.3	length	23
4.25.1.4	sourcePort	23
4.26	zergPacket Struct Reference	24
4.26.1	Field Documentation	24
4.26.1.1	destinationId	24
4.26.1.2	sequenceId	24
4.26.1.3	sourceId	24
4.26.1.4	totalLength	24
4.26.1.5	type	24
4.26.1.6	version	24
4.27	ZergUnit Struct Reference	24
4.27.1	Field Documentation	25
4.27.1.1	dupe	25
4.27.1.2	id	25
4.27.1.3	loc	25
4.27.1.4	seen	25
4.27.1.5	status	25

5 File Documentation	27
5.1 README.md File Reference	27
5.2 src/dijkstra/Dijkstra.c File Reference	27
5.2.1 Function Documentation	28
5.2.1.1 Dijkstra_path(const Graph *g, const char *start, const char *end, char ***path)	28
5.2.1.2 Dijkstra_solveMaze(char **mazeFromFile, char **route, size_t hops)	28
5.3 src/dijkstra/Dijkstra.h File Reference	28
5.3.1 Function Documentation	29
5.3.1.1 Dijkstra_path(const Graph *g, const char *start, const char *end, char ***path)	29
5.3.1.2 Dijkstra_solveMaze(char **mazeFromFile, char **route, size_t hops)	30
5.4 src/dpqueue/heap.c File Reference	30
5.4.1 Function Documentation	31
5.4.1.1 heap_print(const pqueue *pq)	31
5.4.1.2 pqueue_create(int(*cmp)(void *, void *))	31
5.4.1.3 pqueue_dequeue(pqueue *pq, void **item)	31
5.4.1.4 pqueue_destroy(pqueue *pq)	31
5.4.1.5 pqueue_enqueue(pqueue *pq, void *item, double priority)	31
5.4.1.6 pqueue_reprioritize(pqueue *pq, void *item, double priority)	31
5.4.1.7 pqueue_search(const pqueue *pq, void *item)	31
5.4.1.8 pqueue_size(const pqueue *pq)	31
5.5 src/dpqueue/pqueue.c File Reference	31
5.5.1 Function Documentation	32
5.5.1.1 heap_print(const pqueue *pq)	32
5.5.1.2 pqueue_create(int(*cmp)(void *, void *))	32
5.5.1.3 pqueue_dequeue(pqueue *pq, void **item)	32
5.5.1.4 pqueue_destroy(pqueue *pq)	32
5.5.1.5 pqueue_enqueue(pqueue *pq, void *item, double priority)	32
5.5.1.6 pqueue_reprioritize(pqueue *pq, void *item, double priority)	32
5.5.1.7 pqueue_search(const pqueue *pq, void *item)	32
5.5.1.8 pqueue_size(const pqueue *pq)	32

5.6	src/dpqueue/pqueue.h File Reference	32
5.6.1	Typedef Documentation	33
5.6.1.1	pqueue	33
5.6.2	Function Documentation	33
5.6.2.1	pqueue_create(int(*cmp)(void *, void *))	33
5.6.2.2	pqueue_dequeue(pqueue *pq, void **item)	33
5.6.2.3	pqueue_destroy(pqueue *pq)	33
5.6.2.4	pqueue_enqueue(pqueue *pq, void *item, double priority)	33
5.6.2.5	pqueue_reprioritize(pqueue *pq, void *item, double priority)	33
5.6.2.6	pqueue_search(const pqueue *pq, void *item)	33
5.6.2.7	pqueue_size(const pqueue *pq)	33
5.7	src/graph/adjlist.c File Reference	34
5.7.1	Macro Definition Documentation	35
5.7.1.1	_XOPEN_SOURCE	35
5.7.2	Typedef Documentation	35
5.7.2.1	edge_	35
5.7.2.2	node_	35
5.7.3	Function Documentation	35
5.7.3.1	Graph_addEdge(Graph *g, const char *from, const char *to, double weight)	35
5.7.3.2	Graph_addNode(Graph *g, const char *name)	35
5.7.3.3	Graph_create(void)	36
5.7.3.4	Graph_deleteEdge(Graph *g, const char *from, const char *to)	36
5.7.3.5	Graph_deleteNode(Graph *g, const char *name)	36
5.7.3.6	Graph_disassemble(Graph *g)	36
5.7.3.7	Graph_getEdgeWeight(const Graph *g, const char *from, const char *to)	37
5.7.3.8	Graph_getNeighbors(const Graph *g, const char *name, char ***neighbors)	37
5.7.3.9	Graph_getNodes(const Graph *g, char ***nodes)	37
5.7.3.10	Graph_isAdjacent(const Graph *g, const char *from, const char *to)	37
5.7.3.11	Graph_print(const Graph *g)	38
5.7.3.12	Graph_updateEdgeWeight(const Graph *g, const char *from, const char *to, double weight)	38

5.8	src/graph/Graph.c File Reference	38
5.8.1	Macro Definition Documentation	40
5.8.1.1	_XOPEN_SOURCE	40
5.8.2	Typedef Documentation	40
5.8.2.1	edge_	40
5.8.2.2	node_	40
5.8.3	Function Documentation	40
5.8.3.1	Graph_addEdge(Graph *g, const char *from, const char *to, double weight)	40
5.8.3.2	Graph_addNode(Graph *g, const char *name)	40
5.8.3.3	Graph_create(void)	41
5.8.3.4	Graph_deleteEdge(Graph *g, const char *from, const char *to)	41
5.8.3.5	Graph_deleteNode(Graph *g, const char *name)	41
5.8.3.6	Graph_disassemble(Graph *g)	41
5.8.3.7	Graph_getEdgeWeight(const Graph *g, const char *from, const char *to)	41
5.8.3.8	Graph_getNeighbors(const Graph *g, const char *name, char ***neighbors)	42
5.8.3.9	Graph_getNodes(const Graph *g, char ***nodes)	42
5.8.3.10	Graph_isAdjacent(const Graph *g, const char *from, const char *to)	42
5.8.3.11	Graph_print(const Graph *g)	43
5.8.3.12	Graph_updateEdgeWeight(const Graph *g, const char *from, const char *to, double weight)	43
5.9	src/graph/Graph.h File Reference	43
5.9.1	Typedef Documentation	44
5.9.1.1	Graph	44
5.9.2	Function Documentation	44
5.9.2.1	Graph_addEdge(Graph *g, const char *from, const char *to, double weight)	44
5.9.2.2	Graph_addNode(Graph *g, const char *name)	45
5.9.2.3	Graph_create(void)	45
5.9.2.4	Graph_deleteEdge(Graph *g, const char *from, const char *to)	45
5.9.2.5	Graph_deleteNode(Graph *g, const char *name)	46
5.9.2.6	Graph_disassemble(Graph *g)	46
5.9.2.7	Graph_getEdgeWeight(const Graph *g, const char *from, const char *to)	46

5.9.2.8	Graph_getNeighbors(const Graph *g, const char *name, char ***neighbors)	46
5.9.2.9	Graph_getNodes(const Graph *g, char ***nodes)	47
5.9.2.10	Graph_isAdjacent(const Graph *g, const char *from, const char *to)	47
5.9.2.11	Graph_print(const Graph *g)	47
5.9.2.12	Graph_updateEdgeWeight(const Graph *g, const char *from, const char *to, double weight)	47
5.10	src/map/hashtable.c File Reference	48
5.10.1	Macro Definition Documentation	49
5.10.1.1	_XOPEN_SOURCE	49
5.10.2	Function Documentation	49
5.10.2.1	hashtable_print(map *m)	49
5.10.2.2	map_create(void)	49
5.10.2.3	map_destroy(map *m)	49
5.10.2.4	map_exists(map *m, const char *key)	49
5.10.2.5	map_insert(map *m, const char *key, double value)	49
5.10.2.6	map_lookup(map *m, const char *key)	50
5.11	src/map/map.c File Reference	50
5.11.1	Macro Definition Documentation	51
5.11.1.1	_XOPEN_SOURCE	51
5.11.2	Function Documentation	51
5.11.2.1	hashtable_print(map *m)	51
5.11.2.2	map_create(void)	51
5.11.2.3	map_destroy(map *m)	51
5.11.2.4	map_exists(map *m, const char *key)	51
5.11.2.5	map_insert(map *m, const char *key, double value)	52
5.11.2.6	map_lookup(map *m, const char *key)	52
5.12	src/map/map.h File Reference	52
5.12.1	Typedef Documentation	53
5.12.1.1	map	53
5.12.2	Function Documentation	53
5.12.2.1	map_create(void)	53

5.12.2.2	map_destroy(map *m)	53
5.12.2.3	map_exists(map *m, const char *key)	54
5.12.2.4	map_insert(map *m, const char *key, double value)	54
5.12.2.5	map_lookup(map *m, const char *key)	54
5.13	src/map/vmap.c File Reference	55
5.13.1	Macro Definition Documentation	56
5.13.1.1	_XOPEN_SOURCE	56
5.13.2	Function Documentation	56
5.13.2.1	vmap_create(void)	56
5.13.2.2	vmap_destroy(vmap *m)	56
5.13.2.3	vmap_exists(vmap *m, const char *key)	56
5.13.2.4	vmap_insert(vmap *m, const char *key, void *value)	56
5.13.2.5	vmap_lookup(vmap *m, const char *key)	57
5.14	src/map/vmap.h File Reference	57
5.14.1	Typedef Documentation	58
5.14.1.1	vmap	58
5.14.2	Function Documentation	58
5.14.2.1	vmap_create(void)	58
5.14.2.2	vmap_destroy(vmap *m)	58
5.14.2.3	vmap_exists(vmap *m, const char *key)	58
5.14.2.4	vmap_insert(vmap *m, const char *key, void *value)	59
5.14.2.5	vmap_lookup(vmap *m, const char *key)	59
5.15	src/zerg/zergProtos.c File Reference	59
5.15.1	Macro Definition Documentation	61
5.15.1.1	_GNU_SOURCE	61
5.15.2	Function Documentation	61
5.15.2.1	checkEntry(char string[16], unsigned int input, zergPacket *packet)	61
5.15.2.2	create_unit(void)	61
5.15.2.3	deleteRoute(ZergUnit **route, char *node, int *count)	61
5.15.2.4	fileCorruption(void)	61

5.15.2.5	hexToDouble(unsigned long long *myLong, unsigned char hex)	61
5.15.2.6	hexToInt(unsigned int *myInt, unsigned char hex)	62
5.15.2.7	hexToShort(unsigned short *myShort, unsigned char hex)	62
5.15.2.8	parseCapture(FILE *psychicCapture, ZergUnit **unit, int *zergCount)	62
5.15.2.9	pickPacketType(FILE *source, FILE *dest, zergPacket *packet)	62
5.15.2.10	print_zergUnit(ZergUnit *z)	62
5.15.2.11	readCommand(FILE *psychicCapture)	62
5.15.2.12	readEthernetPacket(FILE *psychicCapture)	62
5.15.2.13	readGPS(FILE *psychicCapture, ZergUnit *unit)	62
5.15.2.14	readIpv4Packet(FILE *psychicCapture, unsigned int *ipTotalLength)	62
5.15.2.15	readIpv6Packet(FILE *psychicCapture, unsigned int *ipTotalLength)	62
5.15.2.16	readMessage(FILE *psychicCapture, unsigned int payloadLength)	62
5.15.2.17	readPcapHeader(FILE *psychicCapture)	62
5.15.2.18	readPcapPacket(FILE *psychicCapture)	62
5.15.2.19	readStatus(FILE *psychicCapture, unsigned int payloadLength, ZergUnit *unit)	62
5.15.2.20	readUdpPacket(FILE *psychicCapture, unsigned int *udpTotalLength)	62
5.15.2.21	readZerg(FILE *source, FILE *dest)	62
5.15.2.22	readZergPacket(FILE *psychicCapture, unsigned int *udpTotalLength, ZergUnit **unit, int *zergCount)	62
5.15.2.23	rotate3ByteInt(int swap)	62
5.15.2.24	rotateBack(int swap)	62
5.15.2.25	validateHeader(zergPacket *packet)	62
5.15.2.26	writeCommand(FILE *source, FILE *dest)	62
5.15.2.27	writeEtherHeader(FILE *dest)	62
5.15.2.28	writeGPS(FILE *source, FILE *dest)	63
5.15.2.29	writeIpv4Header(FILE *dest, int zergLength)	63
5.15.2.30	writeMessage(FILE *source, FILE *dest)	63
5.15.2.31	writePcapHeader(FILE *dest)	63
5.15.2.32	writePcapPacket(FILE *dest, int zergLength)	63
5.15.2.33	writeStatus(FILE *source, FILE *dest)	63
5.15.2.34	writeUdpHeader(FILE *dest, int zergLength)	63

5.15.2.35	writeZergHeader(FILE *dest, zergPacket *packet)	63
5.15.2.36	Zerg_twoPaths(Graph *zergGraph, ZergUnit **unitList, int *zergCount, int changeLimit)	63
5.15.2.37	zergUnit_distance(ZergUnit *z1, ZergUnit *z2)	63
5.15.3	Variable Documentation	63
5.15.3.1	fscanNum	64
5.15.3.2	zergPayloadSize	64
5.16	src/zerg/zergProtos.h File Reference	64
5.16.1	Function Documentation	65
5.16.1.1	checkEntry(char string[16], unsigned int input, zergPacket *packet)	65
5.16.1.2	create_unit(void)	65
5.16.1.3	decimalDegreesToDMS(double coordinate)	66
5.16.1.4	deleteRoute(ZergUnit **route, char *node, int *count)	66
5.16.1.5	fileCorruption(void)	67
5.16.1.6	hexToDouble(unsigned long long *myLong, unsigned char hex)	67
5.16.1.7	hexToInt(unsigned int *myInt, unsigned char hex)	67
5.16.1.8	hexToShort(unsigned short *myShort, unsigned char hex)	67
5.16.1.9	parseCapture(FILE *psychicCapture, ZergUnit **unit, int *zergCount)	67
5.16.1.10	pickPacketType(FILE *source, FILE *dest, zergPacket *packet)	67
5.16.1.11	print_zergUnit(ZergUnit *z)	67
5.16.1.12	readCommand(FILE *psychicCapture)	67
5.16.1.13	readEthernetPacket(FILE *psychicCapture)	67
5.16.1.14	readGPS(FILE *psychicCapture, ZergUnit *unit)	67
5.16.1.15	readIpv4Packet(FILE *psychicCapture, unsigned int *ipTotalLength)	67
5.16.1.16	readIpv6Packet(FILE *psychicCaputre, unsigned int *ipTotalLength)	67
5.16.1.17	readMessage(FILE *psychicCapture, unsigned int payloadLength)	67
5.16.1.18	readPcapHeader(FILE *psychicCapture)	67
5.16.1.19	readPcapPacket(FILE *psychicCapture)	67
5.16.1.20	readStatus(FILE *psychicCapture, unsigned int payloadLength, ZergUnit *unit)	67
5.16.1.21	readUdpPacket(FILE *psychicCapture, unsigned int *udpTotalLength)	67
5.16.1.22	readZerg(FILE *source, FILE *dest)	67

5.16.1.23 readZergPacket(FILE *psychicCapture, unsigned int *udpTotalLength, ZergUnit **unit, int *zergCount)	68
5.16.1.24 rotate3ByteInt(int swap)	68
5.16.1.25 rotateBack(int swap)	68
5.16.1.26 validateHeader(zergPacket *packet)	68
5.16.1.27 writeCommand(FILE *source, FILE *dest)	68
5.16.1.28 writeEtherHeader(FILE *dest)	68
5.16.1.29 writeGPS(FILE *source, FILE *dest)	68
5.16.1.30 writeIpv4Header(FILE *dest, int zergLength)	68
5.16.1.31 writeMessage(FILE *source, FILE *dest)	68
5.16.1.32 writePcapHeader(FILE *dest)	68
5.16.1.33 writePcapPacket(FILE *dest, int zergLength)	68
5.16.1.34 writeStatus(FILE *source, FILE *dest)	68
5.16.1.35 writeUdpHeader(FILE *dest, int zergLength)	68
5.16.1.36 writeZergHeader(FILE *dest, zergPacket *packet)	68
5.16.1.37 Zerg_twoPaths(Graph *zergGraph, ZergUnit **unitList, int *zergCount, int changeLimit)	68
5.16.1.38 zergUnit_distance(ZergUnit *z1, ZergUnit *z2)	68
5.17 src/zerg/zergStructs.h File Reference	69
5.17.1 Typedef Documentation	70
5.17.1.1 cPayload	70
5.17.1.2 ethernetHeader	70
5.17.1.3 gpsPayload	70
5.17.1.4 ipv4Header	70
5.17.1.5 ipv6Header	70
5.17.1.6 payload	70
5.17.1.7 pcapFileHeader	71
5.17.1.8 pcapPacketHeader	71
5.17.1.9 udpHeader	71
5.17.1.10 zergPacket	71
5.17.1.11 ZergUnit	71
5.18 src/zergmap.c File Reference	71
5.18.1 Function Documentation	71
5.18.1.1 main(int argc, char *argv[])	71

Chapter 1

README

Initial Commit

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_item	??
_map	??
_pqueue	??
_vmap	??
gpsPayload::accuracy	??
gpsPayload::altitude	??
gpsPayload::bearing	??
cPayload	??
edge_	??
entry	??
ethernetHeader	??
gpsPayload	??
graph	??
ipv4Header	??
ipv6Header	??
gpsPayload::latitude	??
gpsPayload::longitude	??
node_	??
cPayload::param2	??
payload	??
pcapFileHeader	??
pcapPacketHeader	??
gpsPayload::speed	??
payload::sSpeed	??
udpHeader	??
zergPacket	??
ZergUnit	??

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

src/ zergmap.c	??
src/dijkstra/ Dijkstra.c	??
src/dijkstra/ Dijkstra.h	??
src/dpqueue/ heap.c	??
src/dpqueue/ pqueue.c	??
src/dpqueue/ pqueue.h	??
src/graph/ adjlist.c	??
src/graph/ Graph.c	??
src/graph/ Graph.h	??
src/map/ hashtable.c	??
src/map/ map.c	??
src/map/ map.h	??
src/map/ vmap.c	??
src/map/ vmap.h	??
src/zerg/ zergProtos.c	??
src/zerg/ zergProtos.h	??
src/zerg/ zergStructs.h	??

Chapter 4

Data Structure Documentation

4.1 `_item` Struct Reference

Data Fields

- void * [data](#)
- double [priority](#)

4.1.1 Field Documentation

4.1.1.1 void * `_item::data`

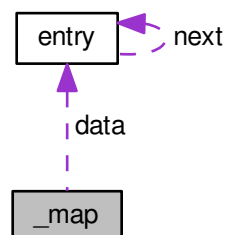
4.1.1.2 double `_item::priority`

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

- `src/dpqueue/heap.c`
- `src/dpqueue/pqueue.c`

4.2 `_map` Struct Reference

Collaboration diagram for `_map`:



Data Fields

- struct [entry](#) * [data](#)
- size_t [size](#)
- size_t [capacity](#)

4.2.1 Field Documentation

4.2.1.1 size_t _map::capacity

4.2.1.2 struct entry * _map::data

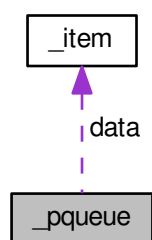
4.2.1.3 size_t _map::size

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

- src/map/[hashtable.c](#)
- src/map/[map.c](#)

4.3 _pqueue Struct Reference

Collaboration diagram for _pqueue:



Data Fields

- struct [_item](#) * [data](#)
- int(* [cmp](#))(void *, void *)
- size_t [size](#)
- size_t [capacity](#)

4.3.1 Field Documentation

4.3.1.1 `size_t _pqueue::capacity`

4.3.1.2 `int(* _pqueue::cmp)(void *, void *)`

4.3.1.3 `struct _item * _pqueue::data`

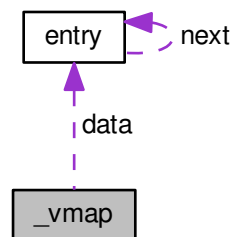
4.3.1.4 `size_t _pqueue::size`

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

- [src/dpqueue/heap.c](#)
- [src/dpqueue/pqueue.c](#)

4.4 _vmap Struct Reference

Collaboration diagram for _vmap:



Data Fields

- struct [entry](#) * `data`
- `size_t` [size](#)
- `size_t` [capacity](#)

4.4.1 Field Documentation

4.4.1.1 `size_t _vmap::capacity`

4.4.1.2 `struct entry* _vmap::data`

4.4.1.3 `size_t _vmap::size`

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

- [src/map/vmap.c](#)

4.5 gpsPayload::accuracy Union Reference

```
#include <zergStructs.h>
```

Data Fields

- float [fAccuracy](#)
- unsigned int [iAccuracy](#)

4.5.1 Field Documentation

4.5.1.1 float gpsPayload::accuracy::fAccuracy

4.5.1.2 unsigned int gpsPayload::accuracy::iAccuracy

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

- src/zerg/[zergStructs.h](#)

4.6 gpsPayload::altitude Union Reference

```
#include <zergStructs.h>
```

Data Fields

- float [fAltitude](#)
- unsigned int [iAltitude](#)

4.6.1 Field Documentation

4.6.1.1 float gpsPayload::altitude::fAltitude

4.6.1.2 unsigned int gpsPayload::altitude::iAltitude

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

- src/zerg/[zergStructs.h](#)

4.7 gpsPayload::bearing Union Reference

```
#include <zergStructs.h>
```

Data Fields

- float [fBearing](#)
- unsigned int [iBearing](#)

4.7.1 Field Documentation

4.7.1.1 float `gpsPayload::bearing::fBearing`

4.7.1.2 unsigned int `gpsPayload::bearing::iBearing`

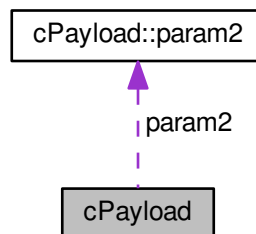
The documentation for this union was generated from the following file:

- `src/zerg/zergStructs.h`

4.8 cPayload Struct Reference

```
#include <zergStructs.h>
```

Collaboration diagram for cPayload:



Data Structures

- union [param2](#)

Data Fields

- unsigned short [command](#)
- unsigned short [param1](#)
- union [cPayload::param2](#) `param2`

4.8.1 Field Documentation

4.8.1.1 unsigned short cPayload::command

4.8.1.2 unsigned short cPayload::param1

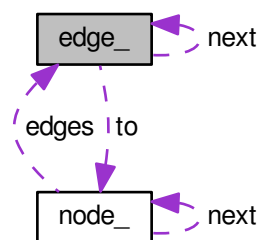
4.8.1.3 union cPayload::param2 cPayload::param2

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

- src/zerg/zergStructs.h

4.9 edge_ Struct Reference

Collaboration diagram for edge_:



Data Fields

- double `weight`
- struct `node_` * `to`
- struct `edge_` * `next`

4.9.1 Field Documentation

4.9.1.1 struct `edge_` * `edge_::next`

4.9.1.2 struct `node_` * `edge_::to`

4.9.1.3 double `edge_::weight`

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

- src/graph/adjlist.c
- src/graph/Graph.c

4.10 entry Struct Reference

Collaboration diagram for entry:



Data Fields

- char * [key](#)
- double [value](#)
- struct [entry](#) * [next](#)
- void * [value](#)

4.10.1 Field Documentation

4.10.1.1 char * entry::key

4.10.1.2 struct entry * entry::next

4.10.1.3 void* entry::value

4.10.1.4 double entry::value

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

- src/map/[hashtable.c](#)
- src/map/[map.c](#)
- src/map/[vmap.c](#)

4.11 ethernetHeader Struct Reference

```
#include <zergStructs.h>
```

Data Fields

- unsigned char [destMac](#) [6]
- unsigned char [sourceMac](#) [6]
- unsigned short int [etherType](#)

4.11.1 Field Documentation

4.11.1.1 unsigned char ethernetHeader::destMac[6]

4.11.1.2 unsigned short int ethernetHeader::etherType

4.11.1.3 unsigned char ethernetHeader::sourceMac[6]

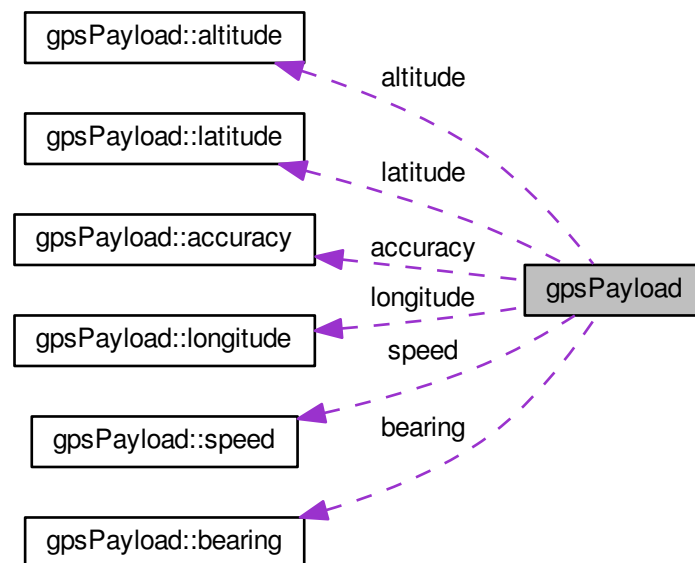
The documentation for this struct was generated from the following file:

- src/zerp/[zergStructs.h](#)

4.12 gpsPayload Struct Reference

```
#include <zergStructs.h>
```

Collaboration diagram for gpsPayload:



Data Structures

- union [accuracy](#)
- union [altitude](#)
- union [bearing](#)
- union [latitude](#)
- union [longitude](#)
- union [speed](#)

Data Fields

- union [gpsPayload::longitude](#) longitude
- union [gpsPayload::latitude](#) latitude
- union [gpsPayload::altitude](#) altitude
- union [gpsPayload::bearing](#) bearing
- union [gpsPayload::speed](#) speed
- union [gpsPayload::accuracy](#) accuracy

4.12.1 Field Documentation

4.12.1.1 union [gpsPayload::accuracy](#) [gpsPayload::accuracy](#)

4.12.1.2 union [gpsPayload::altitude](#) [gpsPayload::altitude](#)

4.12.1.3 union [gpsPayload::bearing](#) [gpsPayload::bearing](#)

4.12.1.4 union [gpsPayload::latitude](#) [gpsPayload::latitude](#)

4.12.1.5 union [gpsPayload::longitude](#) [gpsPayload::longitude](#)

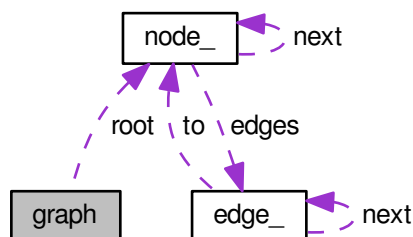
4.12.1.6 union [gpsPayload::speed](#) [gpsPayload::speed](#)

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

- [src/zerg/zergStructs.h](#)

4.13 graph Struct Reference

Collaboration diagram for graph:



Data Fields

- struct [node_](#) * [root](#)

4.13.1 Field Documentation

4.13.1.1 struct [node_](#) * [graph::root](#)

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

- [src/graph/adjlist.c](#)
- [src/graph/Graph.c](#)

4.14 ipv4Header Struct Reference

```
#include <zergStructs.h>
```

Data Fields

- unsigned char [ipHeaderLength](#):4
- unsigned char [version](#):4
- unsigned char [dscp](#)
- unsigned short [ipLength](#)
- unsigned short [id](#)
- unsigned short [flags](#)
- unsigned char [ttl](#)
- unsigned char [protocol](#)
- unsigned short [checksum](#)
- unsigned int [sourceIp](#)
- unsigned int [destIp](#)

4.14.1 Field Documentation

4.14.1.1 unsigned short [ipv4Header::checksum](#)

4.14.1.2 unsigned int [ipv4Header::destIp](#)

4.14.1.3 unsigned char [ipv4Header::dscp](#)

4.14.1.4 unsigned short [ipv4Header::flags](#)

4.14.1.5 unsigned short [ipv4Header::id](#)

4.14.1.6 unsigned char [ipv4Header::ipHeaderLength](#)

4.14.1.7 unsigned short ipv4Header::ipLength

4.14.1.8 unsigned char ipv4Header::protocol

4.14.1.9 unsigned int ipv4Header::sourceIp

4.14.1.10 unsigned char ipv4Header::ttl

4.14.1.11 unsigned char ipv4Header::version

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

- src/zerg/[zergStructs.h](#)

4.15 ipv6Header Struct Reference

```
#include <zergStructs.h>
```

Data Fields

- unsigned char [version](#):4
- unsigned char [trafficClass](#)
- unsigned int [flowLabel](#):20
- unsigned short [payloadLength](#)
- unsigned char [nextHeader](#)
- unsigned char [hopLimit](#)
- unsigned char [source](#) [16]
- unsigned char [destination](#) [16]

4.15.1 Field Documentation

4.15.1.1 unsigned char ipv6Header::destination[16]

4.15.1.2 unsigned int ipv6Header::flowLabel

4.15.1.3 unsigned char ipv6Header::hopLimit

4.15.1.4 unsigned char ipv6Header::nextHeader

4.15.1.5 unsigned short ipv6Header::payloadLength

4.15.1.6 unsigned char ipv6Header::source[16]

4.15.1.7 unsigned char ipv6Header::trafficClass

4.15.1.8 unsigned char ipv6Header::version

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

- src/zerg/[zergStructs.h](#)

4.16 gpsPayload::latitude Union Reference

```
#include <zergStructs.h>
```

Data Fields

- double [dLat](#)
- unsigned long long [iLat](#)

4.16.1 Field Documentation

4.16.1.1 double `gpsPayload::latitude::dLat`

4.16.1.2 unsigned long long `gpsPayload::latitude::iLat`

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

- `src/zerg/zergStructs.h`

4.17 gpsPayload::longitude Union Reference

```
#include <zergStructs.h>
```

Data Fields

- double [dLong](#)
- unsigned long long [iLong](#)

4.17.1 Field Documentation

4.17.1.1 double `gpsPayload::longitude::dLong`

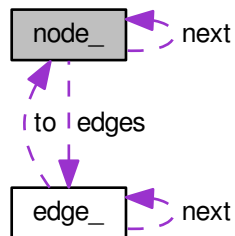
4.17.1.2 unsigned long long `gpsPayload::longitude::iLong`

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

- `src/zerg/zergStructs.h`

4.18 node_ Struct Reference

Collaboration diagram for node_:



Data Fields

- char * [name](#)
- struct [node_](#) * [next](#)
- struct [edge_](#) * [edges](#)

4.18.1 Field Documentation

4.18.1.1 struct [edge_](#) * [node_::edges](#)

4.18.1.2 char * [node_::name](#)

4.18.1.3 struct [node_](#) * [node_::next](#)

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

- [src/graph/adjlist.c](#)
- [src/graph/Graph.c](#)

4.19 cPayload::param2 Union Reference

```
#include <zergStructs.h>
```

Data Fields

- float [fParam2](#)
- int [iParam2](#)
- unsigned int [uiParam2](#)

4.19.1 Field Documentation

4.19.1.1 float cPayload::param2::fParam2

4.19.1.2 int cPayload::param2::iParam2

4.19.1.3 unsigned int cPayload::param2::uiParam2

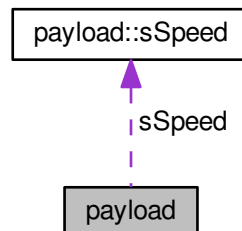
The documentation for this union was generated from the following file:

- src/zerg/zergStructs.h

4.20 payload Struct Reference

```
#include <zergStructs.h>
```

Collaboration diagram for payload:



Data Structures

- union [sSpeed](#)

Data Fields

- unsigned int [currHitPoints](#):24
- unsigned char [armor](#)
- unsigned int [maxHitPoints](#):24
- unsigned char [type](#)
- union [payload::sSpeed](#) [sSpeed](#)

4.20.1 Field Documentation

4.20.1.1 unsigned char payload::armor

4.20.1.2 unsigned int payload::currHitPoints

4.20.1.3 unsigned int payload::maxHitPoints

4.20.1.4 union payload::sSpeed payload::sSpeed

4.20.1.5 unsigned char payload::type

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

- src/zerg/zergStructs.h

4.21 pcapFileHeader Struct Reference

```
#include <zergStructs.h>
```

Data Fields

- unsigned int [fileTypeeld](#)
- unsigned short [majorVersion](#)
- unsigned short [minorVersion](#)
- unsigned int [gmtOffset](#)
- unsigned int [accDelta](#)
- unsigned int [maxLength](#)
- unsigned int [linkLayerType](#)

4.21.1 Field Documentation

4.21.1.1 unsigned int pcapFileHeader::accDelta

4.21.1.2 unsigned int pcapFileHeader::fileTypeeld

4.21.1.3 unsigned int pcapFileHeader::gmtOffset

4.21.1.4 unsigned int pcapFileHeader::linkLayerType

4.21.1.5 unsigned short pcapFileHeader::majorVersion

4.21.1.6 unsigned int pcapFileHeader::maxLength

4.21.1.7 unsigned short pcapFileHeader::minorVersion

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

- src/zerg/zergStructs.h

4.22 pcapPacketHeader Struct Reference

```
#include <zergStructs.h>
```

Data Fields

- unsigned int [unixEpoch](#)
- unsigned int [microEpoch](#)
- unsigned int [lengthOfData](#)
- unsigned int [fullLength](#)

4.22.1 Field Documentation

4.22.1.1 unsigned int pcapPacketHeader::fullLength

4.22.1.2 unsigned int pcapPacketHeader::lengthOfData

4.22.1.3 unsigned int pcapPacketHeader::microEpoch

4.22.1.4 unsigned int pcapPacketHeader::unixEpoch

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

- src/zerg/[zergStructs.h](#)

4.23 gpsPayload::speed Union Reference

```
#include <zergStructs.h>
```

Data Fields

- float [fSpeed](#)
- unsigned int [iSpeed](#)

4.23.1 Field Documentation

4.23.1.1 float gpsPayload::speed::fSpeed

4.23.1.2 unsigned int gpsPayload::speed::iSpeed

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

- src/zerg/[zergStructs.h](#)

4.24 payload::sSpeed Union Reference

```
#include <zergStructs.h>
```

Data Fields

- float [fSpeed](#)
- unsigned int [iSpeed](#)

4.24.1 Field Documentation

4.24.1.1 float payload::sSpeed::fSpeed

4.24.1.2 unsigned int payload::sSpeed::iSpeed

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

- src/zerg/[zergStructs.h](#)

4.25 udpHeader Struct Reference

```
#include <zergStructs.h>
```

Data Fields

- unsigned short [sourcePort](#)
- unsigned short [destPort](#)
- unsigned short [length](#)
- unsigned short [checksum](#)

4.25.1 Field Documentation

4.25.1.1 unsigned short udpHeader::checksum

4.25.1.2 unsigned short udpHeader::destPort

4.25.1.3 unsigned short udpHeader::length

4.25.1.4 unsigned short udpHeader::sourcePort

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

- src/zerg/[zergStructs.h](#)

4.26 zergPacket Struct Reference

```
#include <zergStructs.h>
```

Data Fields

- char [type](#):4
- char [version](#):4
- unsigned int [totalLength](#):24
- unsigned short [sourceId](#)
- unsigned short [destinationId](#)
- unsigned int [sequenceId](#)

4.26.1 Field Documentation

4.26.1.1 unsigned short zergPacket::destinationId

4.26.1.2 unsigned int zergPacket::sequenceId

4.26.1.3 unsigned short zergPacket::sourceId

4.26.1.4 unsigned int zergPacket::totalLength

4.26.1.5 char zergPacket::type

4.26.1.6 char zergPacket::version

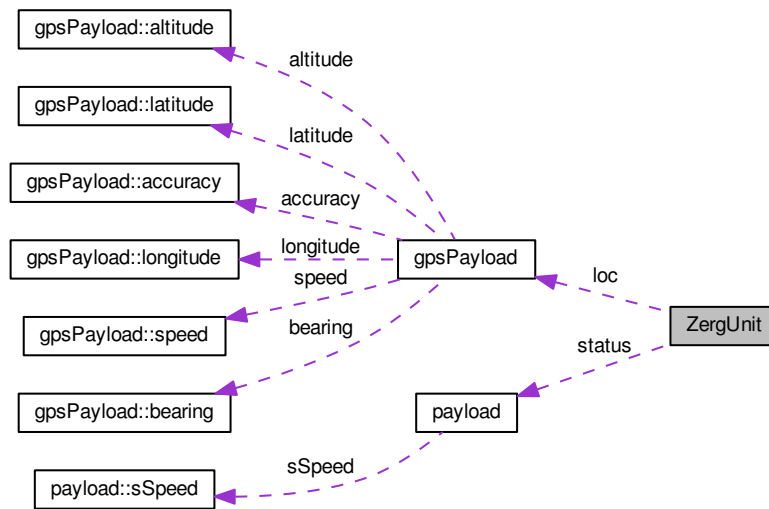
The documentation for this struct was generated from the following file:

- [src/zerg/zergStructs.h](#)

4.27 ZergUnit Struct Reference

```
#include <zergStructs.h>
```


Collaboration diagram for ZergUnit:



Data Fields

- unsigned short `id`
- bool `dupe`
- `payload` * `status`
- `gpsPayload` * `loc`
- int `seen`

4.27.1 Field Documentation

4.27.1.1 bool `ZergUnit::dupe`

4.27.1.2 unsigned short `ZergUnit::id`

4.27.1.3 `gpsPayload` * `ZergUnit::loc`

4.27.1.4 int `ZergUnit::seen`

4.27.1.5 `payload` * `ZergUnit::status`

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

- `src/zerg/zergStructs.h`

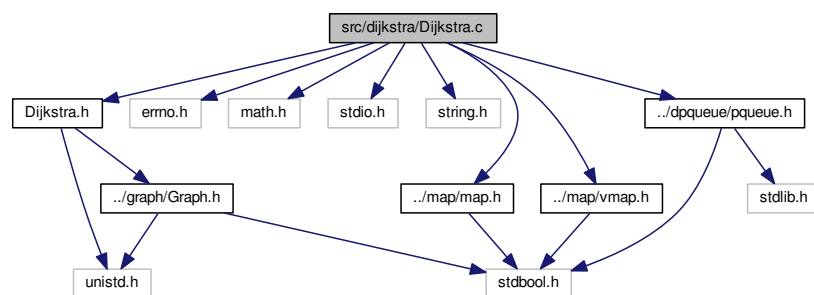
Chapter 5

File Documentation

5.1 README.md File Reference

5.2 src/dijkstra/Dijkstra.c File Reference

```
#include "Dijkstra.h"
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include "../map/map.h"
#include "../map/vmap.h"
#include "../dpqueue/pqueue.h"
Include dependency graph for Dijkstra.c:
```



Functions

- `ssize_t Dijkstra_path (const Graph *g, const char *start, const char *end, char ***path)`
Find path between start and end.
- `void Dijkstra_solveMaze (char **mazeFromFile, char **route, size_t hops)`
Changes 2d array of the maze and changes the route to show the solved maze.

5.2.1 Function Documentation

5.2.1.1 `ssize_t Dijkstra_path (const Graph * g, const char * start, const char * end, char *** path)`

Find path between start and end.

Parameters

<i>g</i>	Graph to traverse
<i>start</i>	starting node in graph
<i>end</i>	ending node in graph
<i>path</i>	output parameter which is filled with shortest start-to-end path

Returns

number of nodes in path (negative if error or disconnected)

5.2.1.2 `void Dijkstra_solveMaze (char ** mazeFromFile, char ** route, size_t hops)`

Changes 2d array of the maze and changes the route to show the solved maze.

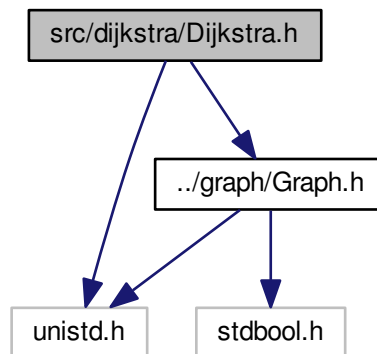
Parameters

<i>mazeFromFile</i>	2d array of maze
<i>route</i>	Nodes in graph that lead to end
<i>hops</i>	Number of hops for tracking in loop

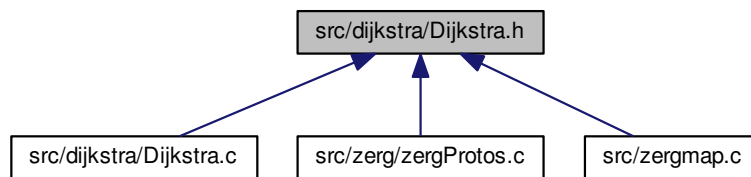
5.3 src/dijkstra/Dijkstra.h File Reference

```
#include <unistd.h>
#include "../graph/Graph.h"
```

Include dependency graph for Dijkstra.h:



This graph shows which files directly or indirectly include this file:



Functions

- `ssize_t Dijkstra_path` (const `Graph` *`g`, const char *`start`, const char *`end`, char ***`path`)
Find path between start and end.
- `void Dijkstra_solveMaze` (char **`mazeFromFile`, char **`route`, size_t `hops`)
Changes 2d array of the maze and changes the route to show the solved maze.

5.3.1 Function Documentation

5.3.1.1 `ssize_t Dijkstra_path (const Graph * g, const char * start, const char * end, char *** path)`

Find path between start and end.

Parameters

<i>g</i>	Graph to traverse
<i>start</i>	starting node in graph
<i>end</i>	ending node in graph
<i>path</i>	output parameter which is filled with shortest start-to-end path

Returns

number of nodes in path (negative if error or disconnected)

5.3.1.2 void Dijkstra_solveMaze (char ** *mazeFromFile*, char ** *route*, size_t *hops*)

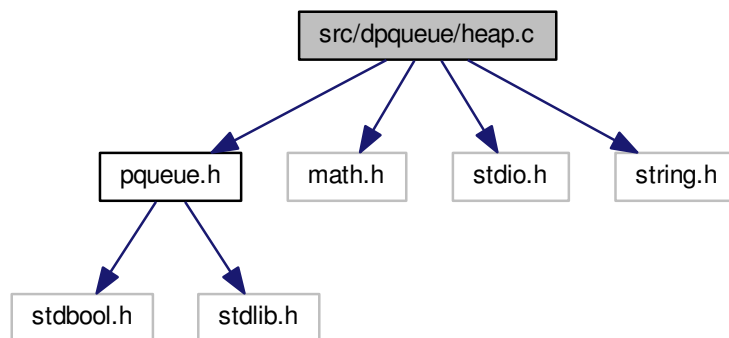
Changes 2d array of the maze and changes the route to show the solved maze.

Parameters

<i>mazeFromFile</i>	2d array of maze
<i>route</i>	Nodes in graph that lead to end
<i>hops</i>	Number of hops for tracking in loop

5.4 src/dpqueue/heap.c File Reference

```
#include "pqueue.h"
#include <math.h>
#include <stdio.h>
#include <string.h>
Include dependency graph for heap.c:
```

**Data Structures**

- [struct _item](#)
- [struct _pqueue](#)

Functions

- `pqueue * pqueue_create` (`int(*cmp)(void *, void *)`)
- `size_t pqueue_size` (`const pqueue *pq`)
- `bool pqueue_enqueue` (`pqueue *pq, void *item, double priority`)
- `bool pqueue_reprioritize` (`pqueue *pq, void *item, double priority`)
- `double pqueue_dequeue` (`pqueue *pq, void **item`)
- `void * pqueue_search` (`const pqueue *pq, void *item`)
- `void pqueue_destroy` (`pqueue *pq`)
- `void heap_print` (`const pqueue *pq`)

5.4.1 Function Documentation

5.4.1.1 `void heap_print (const pqueue * pq)`

5.4.1.2 `pqueue* pqueue_create (int(*)(void *, void *) cmp)`

5.4.1.3 `double pqueue_dequeue (pqueue * pq, void ** item)`

5.4.1.4 `void pqueue_destroy (pqueue * pq)`

5.4.1.5 `bool pqueue_enqueue (pqueue * pq, void * item, double priority)`

5.4.1.6 `bool pqueue_reprioritize (pqueue * pq, void * item, double priority)`

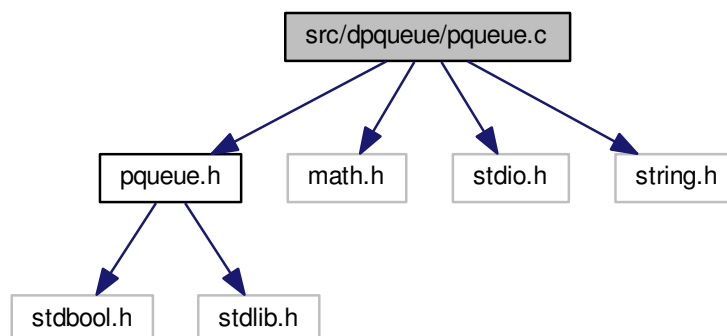
5.4.1.7 `void* pqueue_search (const pqueue * pq, void * item)`

5.4.1.8 `size_t pqueue_size (const pqueue * pq)`

5.5 src/dpqueue/pqueue.c File Reference

```
#include "pqueue.h"
#include <math.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for pqueue.c:



Data Structures

- struct [_item](#)
- struct [_pqueue](#)

Functions

- [pqueue * pqueue_create](#) (int(*cmp)(void *, void *))
- [size_t pqueue_size](#) (const [pqueue](#) *pq)
- [bool pqueue_enqueue](#) ([pqueue](#) *pq, void *item, double priority)
- [bool pqueue_reprioritize](#) ([pqueue](#) *pq, void *item, double priority)
- [double pqueue_dequeue](#) ([pqueue](#) *pq, void **item)
- [void * pqueue_search](#) (const [pqueue](#) *pq, void *item)
- [void pqueue_destroy](#) ([pqueue](#) *pq)
- [void heap_print](#) (const [pqueue](#) *pq)

5.5.1 Function Documentation

5.5.1.1 void heap_print (const [pqueue](#) * *pq*)

5.5.1.2 [pqueue*](#) pqueue_create (int(*)(void *, void *) *cmp*)

5.5.1.3 double pqueue_dequeue ([pqueue](#) * *pq*, void ** *item*)

5.5.1.4 void pqueue_destroy ([pqueue](#) * *pq*)

5.5.1.5 bool pqueue_enqueue ([pqueue](#) * *pq*, void * *item*, double *priority*)

5.5.1.6 bool pqueue_reprioritize ([pqueue](#) * *pq*, void * *item*, double *priority*)

5.5.1.7 void* pqueue_search (const [pqueue](#) * *pq*, void * *item*)

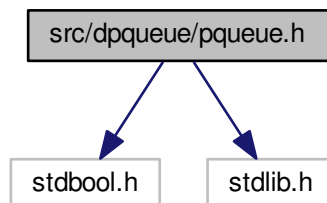
5.5.1.8 size_t pqueue_size (const [pqueue](#) * *pq*)

5.6 src/dpqueue/pqueue.h File Reference

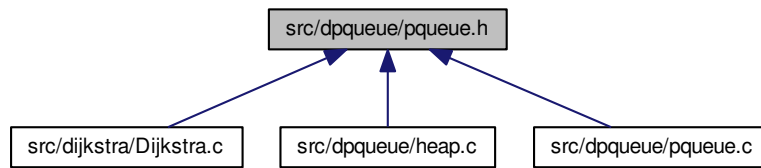
```
#include <stdbool.h>
```

```
#include <stdlib.h>
```

Include dependency graph for pqueue.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct `_pqueue` `pqueue`

Functions

- `pqueue *` `pqueue_create` (`int`(`*``cmp`)(`void *`, `void *`))
- `size_t` `pqueue_size` (`const` `pqueue *``pq`)
- `bool` `pqueue_enqueue` (`pqueue *``pq`, `void *``item`, `double` `priority`)
- `bool` `pqueue_reprioritize` (`pqueue *``pq`, `void *``item`, `double` `priority`)
- `double` `pqueue_dequeue` (`pqueue *``pq`, `void **``item`)
- `void *` `pqueue_search` (`const` `pqueue *``pq`, `void *``item`)
- `void` `pqueue_destroy` (`pqueue *``pq`)

5.6.1 Typedef Documentation

5.6.1.1 typedef struct `_pqueue` `pqueue`

5.6.2 Function Documentation

5.6.2.1 `pqueue*` `pqueue_create` (`int`(`*`)(`void *`, `void *`) `cmp`)

5.6.2.2 `double` `pqueue_dequeue` (`pqueue *` `pq`, `void **` `item`)

5.6.2.3 `void` `pqueue_destroy` (`pqueue *` `pq`)

5.6.2.4 `bool` `pqueue_enqueue` (`pqueue *` `pq`, `void *` `item`, `double` `priority`)

5.6.2.5 `bool` `pqueue_reprioritize` (`pqueue *` `pq`, `void *` `item`, `double` `priority`)

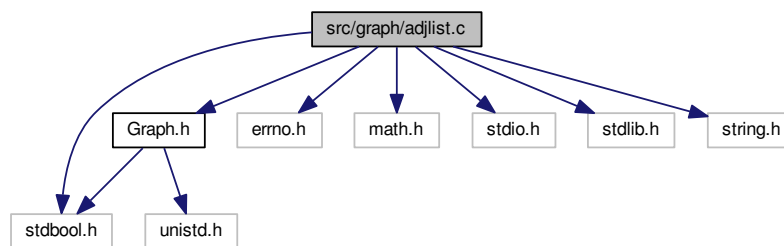
5.6.2.6 `void*` `pqueue_search` (`const` `pqueue *` `pq`, `void *` `item`)

5.6.2.7 `size_t` `pqueue_size` (`const` `pqueue *` `pq`)

5.7 src/graph/adjlist.c File Reference

```
#include "Graph.h"
#include <errno.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for adjlist.c:



Data Structures

- struct [edge_](#)
- struct [node_](#)
- struct [graph](#)

Macros

- `#define _XOPEN_SOURCE 500`

Typedefs

- typedef struct [edge_](#) [edge_](#)
- typedef struct [node_](#) [node_](#)

Functions

- [Graph](#) * [Graph_create](#) (void)
Create an empty graph structure.
- bool [Graph_addNode](#) ([Graph](#) *g, const char *name)
Adds a node to the graph (does not add duplicates)
- bool [Graph_addEdge](#) ([Graph](#) *g, const char *from, const char *to, double weight)
Adds an edge to the graph (does not add duplicates)
- bool [Graph_isAdjacent](#) (const [Graph](#) *g, const char *from, const char *to)
Checks if two nodes are adjacent.
- ssize_t [Graph_getNodes](#) (const [Graph](#) *g, char ***nodes)

- provide list of nodes of a graph*
- ssize_t [Graph_getNeighbors](#) (const [Graph](#) *g, const char *name, char ***neighbors)
- provide list of neighbor's names for a given node*
- double [Graph_getEdgeWeight](#) (const [Graph](#) *g, const char *from, const char *to)
- Provide edge weight between two nodes.*
- bool [Graph_updateEdgeWeight](#) (const [Graph](#) *g, const char *from, const char *to, double weight)
- Update edge weight between two nodes.*
- void [Graph_deleteNode](#) ([Graph](#) *g, const char *name)
- Remove a node from the graph.*
- void [Graph_deleteEdge](#) ([Graph](#) *g, const char *from, const char *to)
- Remove an edge from the graph.*
- void [Graph_print](#) (const [Graph](#) *g)
- Prints graph to stdout.*
- void [Graph_disassemble](#) ([Graph](#) *g)
- Destroy the graph scaffolding without affecting the underlying data.*

5.7.1 Macro Definition Documentation

5.7.1.1 #define _XOPEN_SOURCE 500

5.7.2 Typedef Documentation

5.7.2.1 typedef struct edge_ edge_

5.7.2.2 typedef struct node_ node_

5.7.3 Function Documentation

5.7.3.1 bool [Graph_addEdge](#) ([Graph](#) * g, const char * from, const char * to, double weight)

Adds an edge to the graph (does not add duplicates)

Parameters

<i>g</i>	Graph to modify
<i>from</i>	Name of source node
<i>to</i>	Name of destination node
<i>weight</i>	Cost of the edge

Returns

true for successful add

5.7.3.2 bool [Graph_addNode](#) ([Graph](#) * g, const char * name)

Adds a node to the graph (does not add duplicates)

Parameters

<i>g</i>	Graph to modify
<i>name</i>	Name of new node

Returns

true for successful add

5.7.3.3 Graph* Graph_create (void)

Create an empty graph structure.

Returns

the graph structure, or NULL on error

5.7.3.4 void Graph_deleteEdge (Graph * g, const char * from, const char * to)

Remove an edge from the graph.

Parameters

<i>g</i>	Graph to alter
<i>from</i>	Starting node of edge
<i>to</i>	Ending node of edge

5.7.3.5 void Graph_deleteNode (Graph * g, const char * name)

Remove a node from the graph.

Parameters

<i>g</i>	Graph to alter
<i>name</i>	Name of node to remove

5.7.3.6 void Graph_disassemble (Graph * g)

Destroy the graph scaffolding without affecting the underlying data.

Parameters

<i>g</i>	Graph to disassemble
----------	----------------------

5.7.3.7 `double Graph_getEdgeWeight (const Graph * g, const char * from, const char * to)`

Provide edge weight between two nodes.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Starting node
<i>to</i>	neighbor node

Returns

weight of edge (NAN if edge does not exist)

5.7.3.8 `ssize_t Graph_getNeighbors (const Graph * g, const char * name, char *** neighbors)`

provide list of neighbor's names for a given node

Parameters

<i>g</i>	Graph to inspect
<i>name</i>	node's name to find neighbors for
<i>neighbors</i>	input parameter to store array of neighbors' names

Returns

number of neighbors found (-1 for error)

5.7.3.9 `ssize_t Graph_getNodes (const Graph * g, char *** nodes)`

provide list of nodes of a graph

Parameters

<i>g</i>	Graph to inspect
<i>nodes</i>	input parameter to store array of nodes' names

Returns

number of nodes found (-1 for error)

5.7.3.10 `bool Graph_isAdjacent (const Graph * g, const char * from, const char * to)`

Checks if two nodes are adjacent.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Name of source node
<i>to</i>	Name of destination node

Returns

True if nodes are adjacent, false otherwise

5.7.3.11 void Graph_print (const Graph * *g*)

Prints graph to stdout.

Parameters

<i>g</i>	Graph to print
----------	----------------

5.7.3.12 bool Graph_updateEdgeWeight (const Graph * *g*, const char * *from*, const char * *to*, double *weight*)

Update edge weight between two nodes.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Starting node
<i>to</i>	neighbor node
<i>weight</i>	new weight of edge

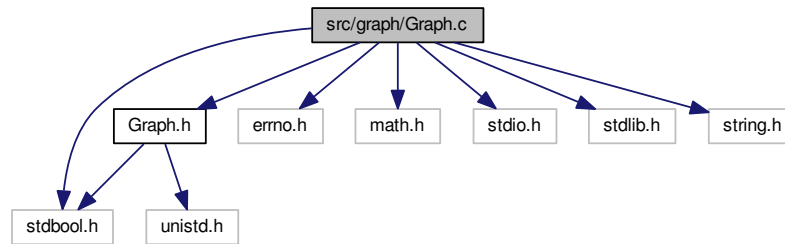
Returns

True if successfully updated, False if failed to update

5.8 src/graph/Graph.c File Reference

```
#include "Graph.h"
#include <errno.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for Graph.c:



Data Structures

- struct [edge_](#)
- struct [node_](#)
- struct [graph](#)

Macros

- `#define _XOPEN_SOURCE 500`

Typedefs

- typedef struct [edge_](#) [edge_](#)
- typedef struct [node_](#) [node_](#)

Functions

- [Graph *](#) [Graph_create](#) (void)
Create an empty graph structure.
- bool [Graph_addNode](#) ([Graph *](#)*g*, const char **name*)
Adds a node to the graph (does not add duplicates)
- bool [Graph_addEdge](#) ([Graph *](#)*g*, const char **from*, const char **to*, double *weight*)
Adds an edge to the graph (does not add duplicates)
- bool [Graph_isAdjacent](#) (const [Graph *](#)*g*, const char **from*, const char **to*)
Checks if two nodes are adjacent.
- ssize_t [Graph_getNodes](#) (const [Graph *](#)*g*, char ****nodes*)
provide list of nodes of a graph
- ssize_t [Graph_getNeighbors](#) (const [Graph *](#)*g*, const char **name*, char ****neighbors*)
provide list of neighbor's names for a given node
- double [Graph_getEdgeWeight](#) (const [Graph *](#)*g*, const char **from*, const char **to*)
Provide edge weight between two nodes.
- bool [Graph_updateEdgeWeight](#) (const [Graph *](#)*g*, const char **from*, const char **to*, double *weight*)
Update edge weight between two nodes.
- void [Graph_deleteNode](#) ([Graph *](#)*g*, const char **name*)

Remove a node from the graph.

- void `Graph_deleteEdge` (`Graph *g`, const char *from, const char *to)

Remove an edge from the graph.

- void `Graph_print` (const `Graph *g`)

Prints graph to stdout.

- void `Graph_disassemble` (`Graph *g`)

Destroy the graph scaffolding without affecting the underlying data.

5.8.1 Macro Definition Documentation

5.8.1.1 `#define _XOPEN_SOURCE 500`

5.8.2 Typedef Documentation

5.8.2.1 `typedef struct edge_ edge_`

5.8.2.2 `typedef struct node_ node_`

5.8.3 Function Documentation

5.8.3.1 `bool Graph_addEdge (Graph * g, const char * from, const char * to, double weight)`

Adds an edge to the graph (does not add duplicates)

Parameters

<i>g</i>	Graph to modify
<i>from</i>	Name of source node
<i>to</i>	Name of destination node
<i>weight</i>	Cost of the edge

Returns

true for successful add

5.8.3.2 `bool Graph_addNode (Graph * g, const char * name)`

Adds a node to the graph (does not add duplicates)

Parameters

<i>g</i>	Graph to modify
<i>name</i>	Name of new node

Returns

true for successful add

5.8.3.3 Graph* Graph_create (void)

Create an empty graph structure.

Returns

the graph structure, or NULL on error

5.8.3.4 void Graph_deleteEdge (Graph * g, const char * from, const char * to)

Remove an edge from the graph.

Parameters

<i>g</i>	Graph to alter
<i>from</i>	Starting node of edge
<i>to</i>	Ending node of edge

5.8.3.5 void Graph_deleteNode (Graph * g, const char * name)

Remove a node from the graph.

Parameters

<i>g</i>	Graph to alter
<i>name</i>	Name of node to remove

5.8.3.6 void Graph_disassemble (Graph * g)

Destroy the graph scaffolding without affecting the underlying data.

Parameters

<i>g</i>	Graph to disassemble
----------	----------------------

5.8.3.7 double Graph_getEdgeWeight (const Graph * g, const char * from, const char * to)

Provide edge weight between two nodes.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Starting node
<i>to</i>	neighbor node

Returns

weight of edge (NAN if edge does not exist)

5.8.3.8 `ssize_t Graph_getNeighbors (const Graph * g, const char * name, char *** neighbors)`

provide list of neighbor's names for a given node

Parameters

<i>g</i>	Graph to inspect
<i>name</i>	node's name to find neighbors for
<i>neighbors</i>	input parameter to store array of neighbors' names

Returns

number of neighbors found (-1 for error)

5.8.3.9 `ssize_t Graph_getNodes (const Graph * g, char *** nodes)`

provide list of nodes of a graph

Parameters

<i>g</i>	Graph to inspect
<i>nodes</i>	input parameter to store array of nodes' names

Returns

number of nodes found (-1 for error)

5.8.3.10 `bool Graph_isAdjacent (const Graph * g, const char * from, const char * to)`

Checks if two nodes are adjacent.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Name of source node
<i>to</i>	Name of destination node

Returns

True if nodes are adjacent, false otherwise

5.8.3.11 void Graph_print (const Graph * g)

Prints graph to stdout.

Parameters

<i>g</i>	Graph to print
----------	----------------

5.8.3.12 bool Graph_updateEdgeWeight (const Graph * g, const char * from, const char * to, double weight)

Update edge weight between two nodes.

Parameters

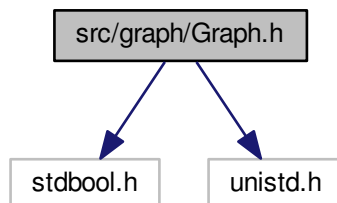
<i>g</i>	Graph to inspect
<i>from</i>	Starting node
<i>to</i>	neighbor node
<i>weight</i>	new weight of edge

Returns

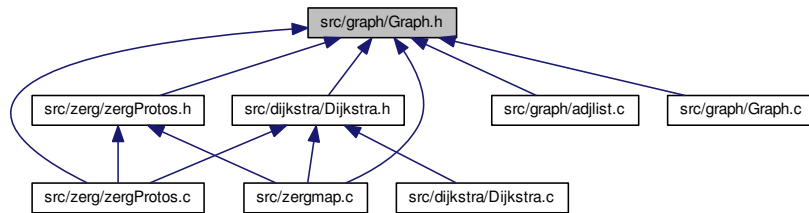
True if successfully updated, False if failed to update

5.9 src/graph/Graph.h File Reference

```
#include <stdbool.h>
#include <unistd.h>
Include dependency graph for Graph.h:
```



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct [graph](#) [Graph](#)

Functions

- [Graph](#) * [Graph_create](#) (void)
Create an empty graph structure.
- bool [Graph_addNode](#) ([Graph](#) *g, const char *name)
Adds a node to the graph (does not add duplicates)
- bool [Graph_addEdge](#) ([Graph](#) *g, const char *from, const char *to, double weight)
Adds an edge to the graph (does not add duplicates)
- bool [Graph_isAdjacent](#) (const [Graph](#) *g, const char *from, const char *to)
Checks if two nodes are adjacent.
- ssize_t [Graph_getNodes](#) (const [Graph](#) *g, char ***nodes)
provide list of nodes of a graph
- ssize_t [Graph_getNeighbors](#) (const [Graph](#) *g, const char *name, char ***neighbors)
provide list of neighbor's names for a given node
- double [Graph_getEdgeWeight](#) (const [Graph](#) *g, const char *from, const char *to)
Provide edge weight between two nodes.
- bool [Graph_updateEdgeWeight](#) (const [Graph](#) *g, const char *from, const char *to, double weight)
Update edge weight between two nodes.
- void [Graph_deleteNode](#) ([Graph](#) *g, const char *name)
Remove a node from the graph.
- void [Graph_deleteEdge](#) ([Graph](#) *g, const char *from, const char *to)
Remove an edge from the graph.
- void [Graph_print](#) (const [Graph](#) *g)
Prints graph to stdout.
- void [Graph_disassemble](#) ([Graph](#) *g)
Destroy the graph scaffolding without affecting the underlying data.

5.9.1 Typedef Documentation

5.9.1.1 typedef struct [graph](#) [Graph](#)

5.9.2 Function Documentation

5.9.2.1 bool [Graph_addEdge](#) ([Graph](#) * g, const char * from, const char * to, double weight)

Adds an edge to the graph (does not add duplicates)

Parameters

<i>g</i>	Graph to modify
<i>from</i>	Name of source node
<i>to</i>	Name of destination node
<i>weight</i>	Cost of the edge

Returns

true for successful add

5.9.2.2 bool Graph_addNode (Graph * *g*, const char * *name*)

Adds a node to the graph (does not add duplicates)

Parameters

<i>g</i>	Graph to modify
<i>name</i>	Name of new node

Returns

true for successful add

5.9.2.3 Graph* Graph_create (void)

Create an empty graph structure.

Returns

the graph structure, or NULL on error

5.9.2.4 void Graph_deleteEdge (Graph * *g*, const char * *from*, const char * *to*)

Remove an edge from the graph.

Parameters

<i>g</i>	Graph to alter
<i>from</i>	Starting node of edge
<i>to</i>	Ending node of edge

5.9.2.5 void Graph_deleteNode (Graph * *g*, const char * *name*)

Remove a node from the graph.

Parameters

<i>g</i>	Graph to alter
<i>name</i>	Name of node to remove

5.9.2.6 void Graph_disassemble (Graph * *g*)

Destroy the graph scaffolding without affecting the underlying data.

Parameters

<i>g</i>	Graph to disassemble
----------	----------------------

5.9.2.7 double Graph_getEdgeWeight (const Graph * *g*, const char * *from*, const char * *to*)

Provide edge weight between two nodes.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Starting node
<i>to</i>	neighbor node

Returns

weight of edge (NAN if edge does not exist)

5.9.2.8 ssize_t Graph_getNeighbors (const Graph * *g*, const char * *name*, char *** *neighbors*)

provide list of neighbor's names for a given node

Parameters

<i>g</i>	Graph to inspect
<i>name</i>	node's name to find neighbors for
<i>neighbors</i>	input parameter to store array of neighbors' names

Returns

number of neighbors found (-1 for error)

5.9.2.9 `ssize_t Graph_getNodes (const Graph * g, char *** nodes)`

provide list of nodes of a graph

Parameters

<i>g</i>	Graph to inspect
<i>nodes</i>	input parameter to store array of nodes' names

Returns

number of nodes found (-1 for error)

5.9.2.10 `bool Graph_isAdjacent (const Graph * g, const char * from, const char * to)`

Checks if two nodes are adjacent.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Name of source node
<i>to</i>	Name of destination node

Returns

True if nodes are adjacent, false otherwise

5.9.2.11 `void Graph_print (const Graph * g)`

Prints graph to stdout.

Parameters

<i>g</i>	Graph to print
----------	----------------

5.9.2.12 `bool Graph_updateEdgeWeight (const Graph * g, const char * from, const char * to, double weight)`

Update edge weight between two nodes.

Parameters

<i>g</i>	Graph to inspect
<i>from</i>	Starting node
<i>to</i>	neighbor node
<i>weight</i>	new weight of edge

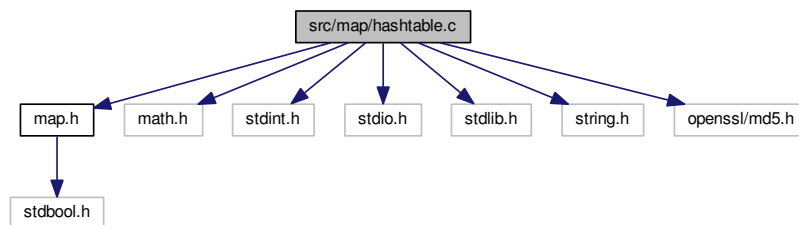
Returns

True if successfully updated, False if failed to update

5.10 src/map/hashtable.c File Reference

```
#include "map.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>
```

Include dependency graph for hashtable.c:



Data Structures

- struct [entry](#)
- struct [_map](#)

Macros

- `#define` [_XOPEN_SOURCE](#) 500

Functions

- `map *` [map_create](#) (void)
Creates an empty map structure.
- `bool` [map_insert](#) (`map` *m, `const char` *key, double value)
Inserts new key and value into map.
- `bool` [map_exists](#) (`map` *m, `const char` *key)
Checks if key exists in map.
- `double` [map_lookup](#) (`map` *m, `const char` *key)
Returns Value of specified key.
- `void` [map_destroy](#) (`map` *m)
Breaks down map and frees memory.
- `void` [hashtable_print](#) (`map` *m)

5.10.1 Macro Definition Documentation

5.10.1.1 `#define _XOPEN_SOURCE 500`

5.10.2 Function Documentation

5.10.2.1 `void hashtable_print (map * m)`

5.10.2.2 `map* map_create (void)`

Creates an empty map structure.

Returns

Pointer to new map in memory

5.10.2.3 `void map_destroy (map * m)`

Breaks down map and frees memory.

Parameters

<i>m</i>	Map to destroy
----------	----------------

5.10.2.4 `bool map_exists (map * m, const char * key)`

Checks if key exists in map.

Parameters

<i>m</i>	Map to inspect
<i>key</i>	Key value to look for in map

Returns

True if found successfully, False if not found

5.10.2.5 `bool map_insert (map * m, const char * key, double value)`

Inserts new key and value into map.

Parameters

<i>m</i>	Map to insert into
<i>key</i>	Key value to add to map
<i>value</i>	Value for specified key

Returns

True if added successfully, False if failed

5.10.2.6 double map_lookup (map * *m*, const char * *key*)

Returns Value of specified key.

Parameters

<i>m</i>	Map to inspect
<i>key</i>	Key value to look for in map

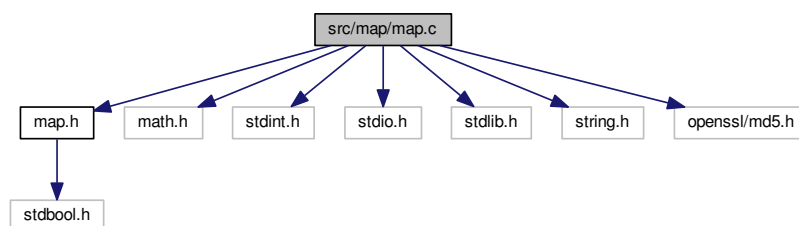
Returns

Value at key specified

5.11 src/map/map.c File Reference

```
#include "map.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>
```

Include dependency graph for map.c:

**Data Structures**

- struct [entry](#)
- struct [_map](#)

Macros

- `#define` [_XOPEN_SOURCE](#) 500

Functions

- `map * map_create (void)`
Creates an empty map structure.
- `bool map_insert (map *m, const char *key, double value)`
Inserts new key and value into map.
- `bool map_exists (map *m, const char *key)`
Checks if key exists in map.
- `double map_lookup (map *m, const char *key)`
Returns Value of specified key.
- `void map_destroy (map *m)`
Breaks down map and frees memory.
- `void hashtable_print (map *m)`

5.11.1 Macro Definition Documentation

5.11.1.1 `#define _XOPEN_SOURCE 500`

5.11.2 Function Documentation

5.11.2.1 `void hashtable_print (map * m)`

5.11.2.2 `map* map_create (void)`

Creates an empty map structure.

Returns

Pointer to new map in memory

5.11.2.3 `void map_destroy (map * m)`

Breaks down map and frees memory.

Parameters

<i>m</i>	Map to destroy
----------	----------------

5.11.2.4 `bool map_exists (map * m, const char * key)`

Checks if key exists in map.

Parameters

<i>m</i>	Map to inspect
<i>key</i>	Key value to look for in map

Returns

True if found successfully, False if not found

5.11.2.5 bool map_insert (map * *m*, const char * *key*, double *value*)

Inserts new key and value into map.

Parameters

<i>m</i>	Map to insert into
<i>key</i>	Key value to add to map
<i>value</i>	Value for specified key

Returns

True if added successfully, False if failed

5.11.2.6 double map_lookup (map * *m*, const char * *key*)

Returns Value of specified key.

Parameters

<i>m</i>	Map to inspect
<i>key</i>	Key value to look for in map

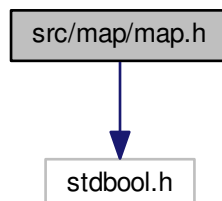
Returns

Value at key specified

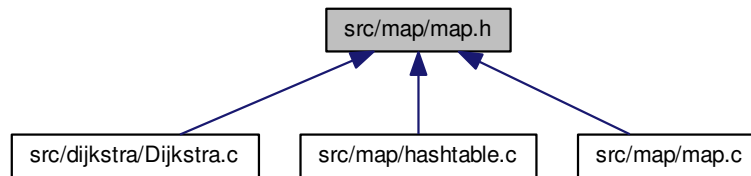
5.12 src/map/map.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for map.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct `_map` `map`

Functions

- `map * map_create (void)`
Creates an empty map structure.
- `bool map_insert (map *m, const char *key, double value)`
Inserts new key and value into map.
- `bool map_exists (map *m, const char *key)`
Checks if key exists in map.
- `double map_lookup (map *m, const char *key)`
Returns Value of specified key.
- `void map_destroy (map *m)`
Breaks down map and frees memory.

5.12.1 Typedef Documentation

5.12.1.1 typedef struct `_map` `map`

5.12.2 Function Documentation

5.12.2.1 `map* map_create (void)`

Creates an empty map structure.

Returns

Pointer to new map in memory

5.12.2.2 `void map_destroy (map * m)`

Breaks down map and frees memory.

Parameters

<i>m</i>	Map to destroy
----------	----------------

5.12.2.3 bool map_exists (map * *m*, const char * *key*)

Checks if key exists in map.

Parameters

<i>m</i>	Map to inspect
<i>key</i>	Key value to look for in map

Returns

True if found successfully, False if not found

5.12.2.4 bool map_insert (map * *m*, const char * *key*, double *value*)

Inserts new key and value into map.

Parameters

<i>m</i>	Map to insert into
<i>key</i>	Key value to add to map
<i>value</i>	Value for specified key

Returns

True if added successfully, False if failed

5.12.2.5 double map_lookup (map * *m*, const char * *key*)

Returns Value of specified key.

Parameters

<i>m</i>	Map to inspect
<i>key</i>	Key value to look for in map

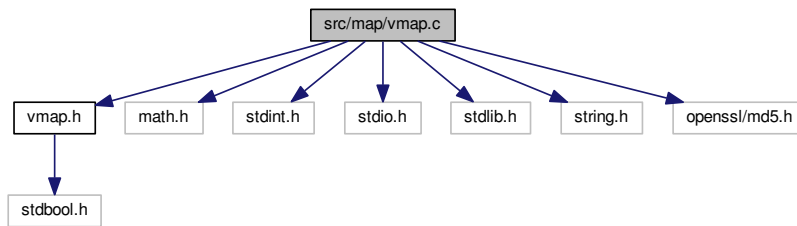
Returns

Value at key specified

5.13 src/map/vmap.c File Reference

```
#include "vmap.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>
```

Include dependency graph for vmmap.c:



Data Structures

- struct [entry](#)
- struct [_vmap](#)

Macros

- `#define` [_XOPEN_SOURCE](#) 500

Functions

- [vmap * vmap_create](#) (void)
Creates an empty vmap structure.
- bool [vmap_insert](#) (vmap *m, const char *key, void *value)
Inserts new key and value into vmap.
- bool [vmap_exists](#) (vmap *m, const char *key)
Checks if key exists in vmap.
- void * [vmap_lookup](#) (vmap *m, const char *key)
Returns Value of specified key.
- void [vmap_destroy](#) (vmap *m)
Breaks down vmap and frees memory.

5.13.1 Macro Definition Documentation

5.13.1.1 `#define _XOPEN_SOURCE 500`

5.13.2 Function Documentation

5.13.2.1 `vmap* vmap_create (void)`

Creates an empty vmap structure.

Returns

Pointer to new vmap in memory

5.13.2.2 `void vmap_destroy (vmap * m)`

Breaks down vmap and frees memory.

Parameters

<i>m</i>	VMap to destroy
----------	-----------------

5.13.2.3 `bool vmap_exists (vmap * m, const char * key)`

Checks if key exists in vmap.

Parameters

<i>m</i>	VMap to inspect
<i>key</i>	Key value to look for in vmap

Returns

True if found successfully, False if not found

5.13.2.4 `bool vmap_insert (vmap * m, const char * key, void * value)`

Inserts new key and value into vmap.

Parameters

<i>m</i>	VMap to insert into
<i>key</i>	Key value to add to vmap
<i>value</i>	Value for specified key

Returns

True if added successfully, False if failed

5.13.2.5 void* vmap_lookup (vmap * m, const char * key)

Returns Value of specified key.

Parameters

<i>m</i>	VMap to inspect
<i>key</i>	Key value to look for in vmap

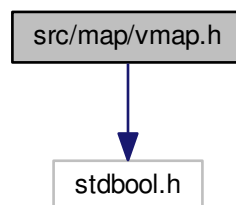
Returns

Value at key specified

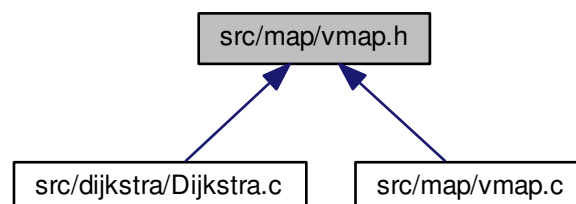
5.14 src/map/vmap.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for vmap.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct `_vmap` `vmap`

Functions

- `vmap * vmap_create` (void)
Creates an empty vmap structure.
- bool `vmap_insert` (`vmap *m`, const char *key, void *value)
Inserts new key and value into vmap.
- bool `vmap_exists` (`vmap *m`, const char *key)
Checks if key exists in vmap.
- void * `vmap_lookup` (`vmap *m`, const char *key)
Returns Value of specified key.
- void `vmap_destroy` (`vmap *m`)
Breaks down vmap and frees memory.

5.14.1 Typedef Documentation

5.14.1.1 typedef struct `_vmap` `vmap`

5.14.2 Function Documentation

5.14.2.1 `vmap* vmap_create (void)`

Creates an empty vmap structure.

Returns

Pointer to new vmap in memory

5.14.2.2 `void vmap_destroy (vmap * m)`

Breaks down vmap and frees memory.

Parameters

<i>m</i>	VMap to destroy
----------	-----------------

5.14.2.3 `bool vmap_exists (vmap * m, const char * key)`

Checks if key exists in vmap.

Parameters

<i>m</i>	VMap to inspect
<i>key</i>	Key value to look for in vmap

Returns

True if found successfully, False if not found

5.14.2.4 `bool vmap_insert (vmap * m, const char * key, void * value)`

Inserts new key and value into vmap.

Parameters

<i>m</i>	VMap to insert into
<i>key</i>	Key value to add to vmap
<i>value</i>	Value for specified key

Returns

True if added successfully, False if failed

5.14.2.5 `void* vmap_lookup (vmap * m, const char * key)`

Returns Value of specified key.

Parameters

<i>m</i>	VMap to inspect
<i>key</i>	Key value to look for in vmap

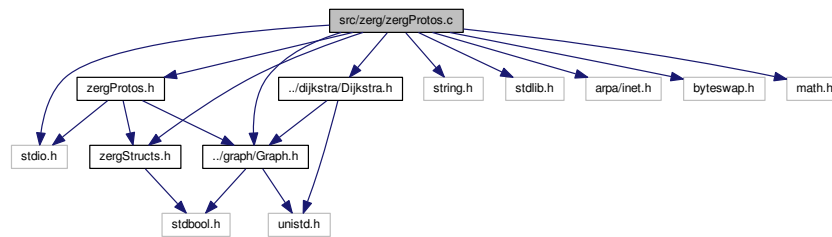
Returns

Value at key specified

5.15 src/zerg/zergProtos.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <byteswap.h>
#include <math.h>
#include "zergProtos.h"
#include "zergStructs.h"
#include "../graph/Graph.h"
#include "../dijkstra/Dijkstra.h"
```

Include dependency graph for zergProtos.c:



Macros

- `#define _GNU_SOURCE`

Functions

- void `readZerg` (FILE *source, FILE *dest)
- void `checkEntry` (char string[16], unsigned int input, `zergPacket` *packet)
- void `pickPacketType` (FILE *source, FILE *dest, `zergPacket` *packet)
- void `writeMessage` (FILE *source, FILE *dest)
- void `writeStatus` (FILE *source, FILE *dest)
- void `writeCommand` (FILE *source, FILE *dest)
- void `writeGPS` (FILE *source, FILE *dest)
- void `writePcapHeader` (FILE *dest)
- void `writePcapPacket` (FILE *dest, int zergLength)
- void `writeEtherHeader` (FILE *dest)
- void `writelpv4Header` (FILE *dest, int zergLength)
- void `writeUdpHeader` (FILE *dest, int zergLength)
- void `writeZergHeader` (FILE *dest, `zergPacket` *packet)
- int `rotate3ByteInt` (int swap)
- int `rotateBack` (int swap)
- void `fileCorruption` (void)
- int `validateHeader` (`zergPacket` *packet)
- void `parseCapture` (FILE *psychicCapture, `ZergUnit` **unit, int *zergCount)
- void `readPcapHeader` (FILE *psychicCapture)
- void `readPcapPacket` (FILE *psychicCapture)
- void `readEthernetPacket` (FILE *psychicCapture)
- void `hexToInt` (unsigned int *myInt, unsigned char hex)
- void `hexToShort` (unsigned short *myShort, unsigned char hex)
- void `hexToDouble` (unsigned long long *myLong, unsigned char hex)
- void `readIpv4Packet` (FILE *psychicCapture, unsigned int *ipTotalLength)
- void `readIpv6Packet` (FILE *psychicCapture, unsigned int *ipTotalLength)
- void `readUdpPacket` (FILE *psychicCapture, unsigned int *udpTotalLength)
- void `readZergPacket` (FILE *psychicCapture, unsigned int *udpTotalLength, `ZergUnit` **unit, int *zergCount)
- void `readMessage` (FILE *psychicCapture, unsigned int payloadLength)
- void `readStatus` (FILE *psychicCapture, unsigned int payloadLength, `ZergUnit` *unit)
- void `readCommand` (FILE *psychicCapture)
- void `readGPS` (FILE *psychicCapture, `ZergUnit` *unit)
- `ZergUnit` * `create_unit` (void)

- Creates and initializes a new [ZergUnit](#).*
- void [print_zergUnit](#) ([ZergUnit](#) *z)
Prints id and GPS information for [ZergUnit](#).
- double [zergUnit_distance](#) ([ZergUnit](#) *z1, [ZergUnit](#) *z2)
Calculates the difference in location between two zerg units.
- char ** [Zerg_twoPaths](#) ([Graph](#) *zergGraph, [ZergUnit](#) **unitList, int *zergCount, int changeLimit)
My attempt to find disjointed node paths between two nodes.
- void [deleteRoute](#) ([ZergUnit](#) **route, char *node, int *count)
Used to delete and shift zergUnit list after deletion.

Variables

- int [zergPayloadSize](#) = 0
- int [fscanNum](#) = 0

5.15.1 Macro Definition Documentation

5.15.1.1 `#define _GNU_SOURCE`

5.15.2 Function Documentation

5.15.2.1 void [checkEntry](#) (char *string*[16], unsigned int *input*, [zergPacket](#) * *packet*)

5.15.2.2 [ZergUnit](#)* [create_unit](#) (void)

Creates and initializes a new [ZergUnit](#).

Returns

Pointer to new [ZergUnit](#)

5.15.2.3 void [deleteRoute](#) ([ZergUnit](#) ** *route*, char * *node*, int * *count*)

Used to delete and shift zergUnit list after deletion.

Parameters

<i>route</i>	zergUnit list that had been traversed
<i>node</i>	Zerg id as a string to be removed
<i>count</i>	Pointer to count to keep track of number of zergUnits

5.15.2.4 void [fileCorruption](#) (void)

5.15.2.5 void [hexToDouble](#) (unsigned long long * *myLong*, unsigned char *hex*)

5.15.2.6 void hexToInt (unsigned int * *myInt*, unsigned char *hex*)

5.15.2.7 void hexToShort (unsigned short * *myShort*, unsigned char *hex*)

5.15.2.8 void parseCapture (FILE * *psychicCapture*, ZergUnit ** *unit*, int * *zergCount*)

5.15.2.9 void pickPacketType (FILE * *source*, FILE * *dest*, zergPacket * *packet*)

5.15.2.10 void print_zergUnit (ZergUnit * *z*)

Prints id and GPS information for [ZergUnit](#).

5.15.2.11 void readCommand (FILE * *psychicCapture*)

5.15.2.12 void readEthernetPacket (FILE * *psychicCapture*)

5.15.2.13 void readGPS (FILE * *psychicCapture*, ZergUnit * *unit*)

5.15.2.14 void readIpv4Packet (FILE * *psychicCapture*, unsigned int * *ipTotalLength*)

5.15.2.15 void readIpv6Packet (FILE * *psychicCapture*, unsigned int * *ipTotalLength*)

5.15.2.16 void readMessage (FILE * *psychicCapture*, unsigned int *payloadLength*)

5.15.2.17 void readPcapHeader (FILE * *psychicCapture*)

5.15.2.18 void readPcapPacket (FILE * *psychicCapture*)

5.15.2.19 void readStatus (FILE * *psychicCapture*, unsigned int *payloadLength*, ZergUnit * *unit*)

5.15.2.20 void readUdpPacket (FILE * *psychicCapture*, unsigned int * *udpTotalLength*)

5.15.2.21 void readZerg (FILE * *source*, FILE * *dest*)

5.15.2.22 void readZergPacket (FILE * *psychicCapture*, unsigned int * *udpTotalLength*, ZergUnit ** *unit*, int * *zergCount*)

5.15.2.23 int rotate3ByteInt (int *swap*)

5.15.2.24 int rotateBack (int *swap*)

5.15.2.25 int validateHeader (zergPacket * *packet*)

5.15.2.26 void writeCommand (FILE * *source*, FILE * *dest*)

5.15.2.27 void writeEtherHeader (FILE * *dest*)

5.15.2.28 void writeGPS (FILE * *source*, FILE * *dest*)

5.15.2.29 void writeIpv4Header (FILE * *dest*, int *zergLength*)

5.15.2.30 void writeMessage (FILE * *source*, FILE * *dest*)

5.15.2.31 void writePcapHeader (FILE * *dest*)

5.15.2.32 void writePcapPacket (FILE * *dest*, int *zergLength*)

5.15.2.33 void writeStatus (FILE * *source*, FILE * *dest*)

5.15.2.34 void writeUdpHeader (FILE * *dest*, int *zergLength*)

5.15.2.35 void writeZergHeader (FILE * *dest*, zergPacket * *packet*)

5.15.2.36 char** Zerg_twoPaths (Graph * *zergGraph*, ZergUnit ** *unitList*, int * *zergCount*, int *changeLimit*)

My attempt to find disjointed node paths between two nodes.

Parameters

<i>zergGraph</i>	Graph of ZergUnits to check
<i>unitList</i>	Array of ZergUnit pointers
<i>zergCount</i>	Pointer to number of ZergUnits in unitList
<i>changeLimit</i>	Used to set how many Zergs can be deleted

Returns

List of node id's that were deleted

5.15.2.37 double zergUnit_distance (ZergUnit * *z1*, ZergUnit * *z2*)

Calculates the difference in location between two zerg units.

Parameters

<i>z1</i>	From Zerg
<i>z2</i>	To Zerg

Returns

Distance as Double

5.15.3 Variable Documentation

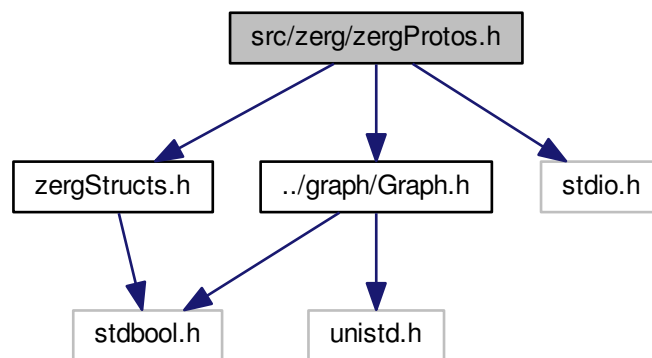
5.15.3.1 `int fscanNum = 0`

5.15.3.2 `int zergPayloadSize = 0`

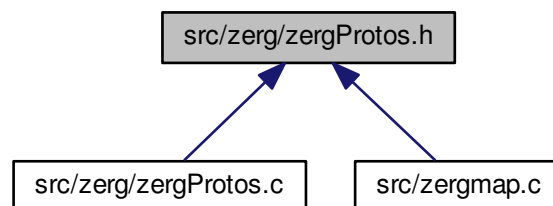
5.16 `src/zerg/zergProtos.h` File Reference

```
#include "zergStructs.h"
#include "../graph/Graph.h"
#include <stdio.h>
```

Include dependency graph for `zergProtos.h`:



This graph shows which files directly or indirectly include this file:



Functions

- void [readZerg](#) (FILE *source, FILE *dest)
- void [checkEntry](#) (char string[16], unsigned int input, [zergPacket](#) *packet)
- void [pickPacketType](#) (FILE *source, FILE *dest, [zergPacket](#) *packet)
- void [writeMessage](#) (FILE *source, FILE *dest)

- void [writeStatus](#) (FILE *source, FILE *dest)
- void [writeCommand](#) (FILE *source, FILE *dest)
- void [writeGPS](#) (FILE *source, FILE *dest)
- void [writePcapHeader](#) (FILE *dest)
- void [writePcapPacket](#) (FILE *dest, int zergLength)
- void [writeEtherHeader](#) (FILE *dest)
- void [writeIpv4Header](#) (FILE *dest, int zergLength)
- void [writeUdpHeader](#) (FILE *dest, int zergLength)
- void [writeZergHeader](#) (FILE *dest, [zergPacket](#) *packet)
- int [rotate3ByteInt](#) (int swap)
- int [rotateBack](#) (int swap)
- void [fileCorruption](#) (void)
- int [validateHeader](#) ([zergPacket](#) *packet)
- [ZergUnit](#) * [create_unit](#) (void)
Creates and initializes a new [ZergUnit](#).
- void [parseCapture](#) (FILE *psychicCapture, [ZergUnit](#) **unit, int *zergCount)
- void [readPcapHeader](#) (FILE *psychicCapture)
- void [readPcapPacket](#) (FILE *psychicCapture)
- void [readEthernetPacket](#) (FILE *psychicCapture)
- void [readIpv4Packet](#) (FILE *psychicCapture, unsigned int *ipTotalLength)
- void [readIpv6Packet](#) (FILE *psychicCapture, unsigned int *ipTotalLength)
- void [readUdpPacket](#) (FILE *psychicCapture, unsigned int *udpTotalLength)
- void [readZergPacket](#) (FILE *psychicCapture, unsigned int *udpTotalLength, [ZergUnit](#) **unit, int *zergCount)
- void [readMessage](#) (FILE *psychicCapture, unsigned int payloadLength)
- void [readStatus](#) (FILE *psychicCapture, unsigned int payloadLength, [ZergUnit](#) *unit)
- void [readCommand](#) (FILE *psychicCapture)
- void [readGPS](#) (FILE *psychicCapture, [ZergUnit](#) *unit)
- void [hexToInt](#) (unsigned int *myInt, unsigned char hex)
- void [hexToShort](#) (unsigned short *myShort, unsigned char hex)
- void [hexToDouble](#) (unsigned long long *myLong, unsigned char hex)
- void [decimalDegreesToDMS](#) (double coordinate)
- double [zergUnit_distance](#) ([ZergUnit](#) *z1, [ZergUnit](#) *z2)
Calculates the difference in location between two zerg units.
- void [print_zergUnit](#) ([ZergUnit](#) *z)
Prints id and GPS information for [ZergUnit](#).
- char ** [Zerg_twoPaths](#) ([Graph](#) *zergGraph, [ZergUnit](#) **unitList, int *zergCount, int changeLimit)
My attempt to find disjointed node paths between two nodes.
- void [deleteRoute](#) ([ZergUnit](#) **route, char *node, int *count)
Used to delete and shift zergUnit list after deletion.

5.16.1 Function Documentation

5.16.1.1 void [checkEntry](#) (char *string*[16], unsigned int *input*, [zergPacket](#) * *packet*)

5.16.1.2 [ZergUnit](#)* [create_unit](#) (void)

Creates and initializes a new [ZergUnit](#).

Returns

Pointer to new [ZergUnit](#)

5.16.1.3 void decimalDegreesToDMS (double *coordinate*)

5.16.1.4 void deleteRoute (ZergUnit ** *route*, char * *node*, int * *count*)

Used to delete and shift zergUnit list after deletion.

Parameters

<i>route</i>	zergUnit list that had been traversed
<i>node</i>	Zerg id as a string to be removed
<i>count</i>	Pointer to count to keep track of number of zergUnits

5.16.1.5 void fileCorruption (void)

5.16.1.6 void hexToDouble (unsigned long long * *myLong*, unsigned char *hex*)

5.16.1.7 void hexToInt (unsigned int * *myInt*, unsigned char *hex*)

5.16.1.8 void hexToShort (unsigned short * *myShort*, unsigned char *hex*)

5.16.1.9 void parseCapture (FILE * *psychicCapture*, ZergUnit ** *unit*, int * *zergCount*)

5.16.1.10 void pickPacketType (FILE * *source*, FILE * *dest*, zergPacket * *packet*)

5.16.1.11 void print_zergUnit (ZergUnit * *z*)

Prints id and GPS information for [ZergUnit](#).

5.16.1.12 void readCommand (FILE * *psychicCapture*)

5.16.1.13 void readEthernetPacket (FILE * *psychicCapture*)

5.16.1.14 void readGPS (FILE * *psychicCapture*, ZergUnit * *unit*)

5.16.1.15 void readIpv4Packet (FILE * *psychicCapture*, unsigned int * *ipTotalLength*)

5.16.1.16 void readIpv6Packet (FILE * *psychicCaputre*, unsigned int * *ipTotalLength*)

5.16.1.17 void readMessage (FILE * *psychicCapture*, unsigned int *payloadLength*)

5.16.1.18 void readPcapHeader (FILE * *psychicCapture*)

5.16.1.19 void readPcapPacket (FILE * *psychicCapture*)

5.16.1.20 void readStatus (FILE * *psychicCapture*, unsigned int *payloadLength*, ZergUnit * *unit*)

5.16.1.21 void readUdpPacket (FILE * *psychicCapture*, unsigned int * *udpTotalLength*)

5.16.1.22 void readZerg (FILE * *source*, FILE * *dest*)

- 5.16.1.23 void readZergPacket (FILE * *psychicCapture*, unsigned int * *udpTotalLength*, ZergUnit ** *unit*, int * *zergCount*)
- 5.16.1.24 int rotate3ByteInt (int *swap*)
- 5.16.1.25 int rotateBack (int *swap*)
- 5.16.1.26 int validateHeader (zergPacket * *packet*)
- 5.16.1.27 void writeCommand (FILE * *source*, FILE * *dest*)
- 5.16.1.28 void writeEtherHeader (FILE * *dest*)
- 5.16.1.29 void writeGPS (FILE * *source*, FILE * *dest*)
- 5.16.1.30 void writeIpv4Header (FILE * *dest*, int *zergLength*)
- 5.16.1.31 void writeMessage (FILE * *source*, FILE * *dest*)
- 5.16.1.32 void writePcapHeader (FILE * *dest*)
- 5.16.1.33 void writePcapPacket (FILE * *dest*, int *zergLength*)
- 5.16.1.34 void writeStatus (FILE * *source*, FILE * *dest*)
- 5.16.1.35 void writeUdpHeader (FILE * *dest*, int *zergLength*)
- 5.16.1.36 void writeZergHeader (FILE * *dest*, zergPacket * *packet*)
- 5.16.1.37 char** Zerg_twoPaths (Graph * *zergGraph*, ZergUnit ** *unitList*, int * *zergCount*, int *changeLimit*)

My attempt to find disjointed node paths between two nodes.

Parameters

<i>zergGraph</i>	Graph of ZergUnits to check
<i>unitList</i>	Array of ZergUnit pointers
<i>zergCount</i>	Pointer to number of ZergUnits in unitList
<i>changeLimit</i>	Used to set how many Zergs can be deleted

Returns

List of node id's that were deleted

- 5.16.1.38 double zergUnit_distance (ZergUnit * *z1*, ZergUnit * *z2*)

Calculates the difference in location between two zerg units.

Parameters

z1	From Zerg
z2	To Zerg

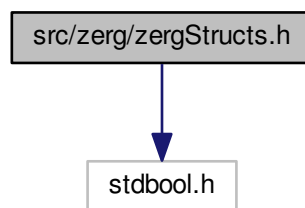
Returns

Distance as Double

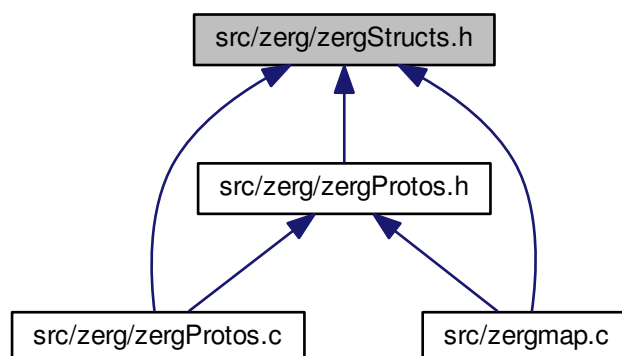
5.17 src/zerg/zergStructs.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for zergStructs.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [zergPacket](#)
- struct [pcapFileHeader](#)
- struct [pcapPacketHeader](#)
- struct [ethernetHeader](#)
- struct [ipv4Header](#)
- struct [ipv6Header](#)
- struct [udpHeader](#)
- struct [payload](#)
- union [payload::sSpeed](#)
- struct [cPayload](#)
- union [cPayload::param2](#)
- struct [gpsPayload](#)
- union [gpsPayload::longitude](#)
- union [gpsPayload::latitude](#)
- union [gpsPayload::altitude](#)
- union [gpsPayload::bearing](#)
- union [gpsPayload::speed](#)
- union [gpsPayload::accuracy](#)
- struct [ZergUnit](#)

Typedefs

- typedef struct [zergPacket](#) [zergPacket](#)
- typedef struct [pcapFileHeader](#) [pcapFileHeader](#)
- typedef struct [pcapPacketHeader](#) [pcapPacketHeader](#)
- typedef struct [ethernetHeader](#) [ethernetHeader](#)
- typedef struct [ipv4Header](#) [ipv4Header](#)
- typedef struct [ipv6Header](#) [ipv6Header](#)
- typedef struct [udpHeader](#) [udpHeader](#)
- typedef struct [payload](#) [payload](#)
- typedef struct [cPayload](#) [cPayload](#)
- typedef struct [gpsPayload](#) [gpsPayload](#)
- typedef struct [ZergUnit](#) [ZergUnit](#)

5.17.1 Typedef Documentation

5.17.1.1 typedef struct [cPayload](#) [cPayload](#)

5.17.1.2 typedef struct [ethernetHeader](#) [ethernetHeader](#)

5.17.1.3 typedef struct [gpsPayload](#) [gpsPayload](#)

5.17.1.4 typedef struct [ipv4Header](#) [ipv4Header](#)

5.17.1.5 typedef struct [ipv6Header](#) [ipv6Header](#)

5.17.1.6 typedef struct [payload](#) [payload](#)

5.17.1.7 typedef struct pcapFileHeader pcapFileHeader

5.17.1.8 typedef struct pcapPacketHeader pcapPacketHeader

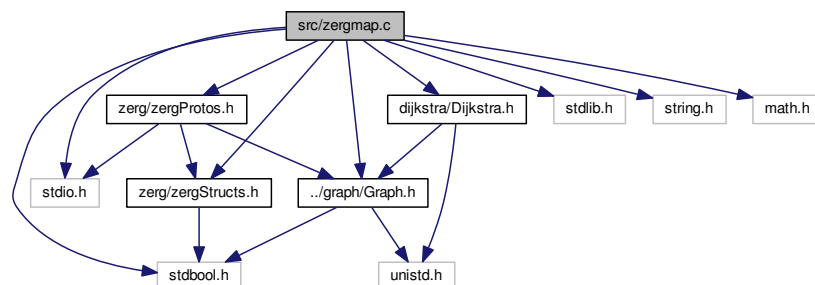
5.17.1.9 typedef struct udpHeader udpHeader

5.17.1.10 typedef struct zergPacket zergPacket

5.17.1.11 typedef struct ZergUnit ZergUnit

5.18 src/zergmap.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "zerg/zergStructs.h"
#include "zerg/zergProtos.h"
#include "graph/Graph.h"
#include "dijkstra/Dijkstra.h"
Include dependency graph for zergmap.c:
```



Functions

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

5.18.1 Function Documentation

5.18.1.1 int main (int argc, char * argv[])

Index

- [_GNU_SOURCE](#)
 - [zergProtos.c, 61](#)
 - [_XOPEN_SOURCE](#)
 - [adjlist.c, 35](#)
 - [Graph.c, 40](#)
 - [hashtable.c, 49](#)
 - [map.c, 51](#)
 - [vmap.c, 56](#)
 - [_item, 7](#)
 - [data, 7](#)
 - [priority, 7](#)
 - [_map, 7](#)
 - [capacity, 8](#)
 - [data, 8](#)
 - [size, 8](#)
 - [_pqueue, 8](#)
 - [capacity, 9](#)
 - [cmp, 9](#)
 - [data, 9](#)
 - [size, 9](#)
 - [_vmap, 9](#)
 - [capacity, 9](#)
 - [data, 9](#)
 - [size, 9](#)
- [accDelta](#)
 - [pcapFileHeader, 21](#)
- [accuracy](#)
 - [gpsPayload, 15](#)
- [adjlist.c](#)
 - [_XOPEN_SOURCE, 35](#)
 - [edge_, 35](#)
 - [Graph_addEdge, 35](#)
 - [Graph_addNode, 35](#)
 - [Graph_create, 36](#)
 - [Graph_deleteEdge, 36](#)
 - [Graph_deleteNode, 36](#)
 - [Graph_disassemble, 36](#)
 - [Graph_getEdgeWeight, 37](#)
 - [Graph_getNeighbors, 37](#)
 - [Graph_getNodes, 37](#)
 - [Graph_isAdjacent, 37](#)
 - [Graph_print, 38](#)
 - [Graph_updateEdgeWeight, 38](#)
 - [node_, 35](#)
- [altitude](#)
 - [gpsPayload, 15](#)
- [armor](#)
 - [payload, 21](#)
- [bearing](#)
 - [gpsPayload, 15](#)
- [cPayload, 11](#)
 - [command, 12](#)
 - [param1, 12](#)
 - [param2, 12](#)
 - [zergStructs.h, 70](#)
- [cPayload::param2, 19](#)
 - [fParam2, 20](#)
 - [iParam2, 20](#)
 - [uiParam2, 20](#)
- [capacity](#)
 - [_map, 8](#)
 - [_pqueue, 9](#)
 - [_vmap, 9](#)
- [checkEntry](#)
 - [zergProtos.c, 61](#)
 - [zergProtos.h, 65](#)
- [checksum](#)
 - [ipv4Header, 16](#)
 - [udpHeader, 23](#)
- [cmp](#)
 - [_pqueue, 9](#)
- [command](#)
 - [cPayload, 12](#)
- [create_unit](#)
 - [zergProtos.c, 61](#)
 - [zergProtos.h, 65](#)
- [currHitPoints](#)
 - [payload, 21](#)
- [dLat](#)
 - [gpsPayload::latitude, 18](#)
- [dLong](#)
 - [gpsPayload::longitude, 18](#)
- [data](#)
 - [_item, 7](#)
 - [_map, 8](#)
 - [_pqueue, 9](#)
 - [_vmap, 9](#)
- [decimalDegreesToDMS](#)
 - [zergProtos.h, 65](#)
- [deleteRoute](#)
 - [zergProtos.c, 61](#)
 - [zergProtos.h, 66](#)
- [destIp](#)
 - [ipv4Header, 16](#)
- [destMac](#)
 - [ethernetHeader, 14](#)

- destPort
 - udpHeader, [23](#)
- destination
 - ipv6Header, [17](#)
- destinationId
 - zergPacket, [24](#)
- Dijkstra.c
 - Dijkstra_path, [28](#)
 - Dijkstra_solveMaze, [28](#)
- Dijkstra.h
 - Dijkstra_path, [29](#)
 - Dijkstra_solveMaze, [30](#)
- Dijkstra_path
 - Dijkstra.c, [28](#)
 - Dijkstra.h, [29](#)
- Dijkstra_solveMaze
 - Dijkstra.c, [28](#)
 - Dijkstra.h, [30](#)
- dscp
 - ipv4Header, [16](#)
- dupe
 - ZergUnit, [25](#)
- edge_, [12](#)
 - adjlist.c, [35](#)
 - Graph.c, [40](#)
 - next, [12](#)
 - to, [12](#)
 - weight, [12](#)
- edges
 - node_, [19](#)
- entry, [13](#)
 - key, [13](#)
 - next, [13](#)
 - value, [13](#)
- etherType
 - ethernetHeader, [14](#)
- ethernetHeader, [13](#)
 - destMac, [14](#)
 - etherType, [14](#)
 - sourceMac, [14](#)
 - zergStructs.h, [70](#)
- fAccuracy
 - gpsPayload::accuracy, [10](#)
- fAltitude
 - gpsPayload::altitude, [10](#)
- fBearing
 - gpsPayload::bearing, [11](#)
- fParam2
 - cPayload::param2, [20](#)
- fSpeed
 - gpsPayload::speed, [22](#)
 - payload::sSpeed, [23](#)
- fileCorruption
 - zergProtos.c, [61](#)
 - zergProtos.h, [67](#)
- fileTypeld
 - pcapFileHeader, [21](#)
- flags
 - ipv4Header, [16](#)
- flowLabel
 - ipv6Header, [17](#)
- fscanNum
 - zergProtos.c, [63](#)
- fullLength
 - pcapPacketHeader, [22](#)
- gmtOffset
 - pcapFileHeader, [21](#)
- gpsPayload, [14](#)
 - accuracy, [15](#)
 - altitude, [15](#)
 - bearing, [15](#)
 - latitude, [15](#)
 - longitude, [15](#)
 - speed, [15](#)
 - zergStructs.h, [70](#)
- gpsPayload::accuracy, [10](#)
 - fAccuracy, [10](#)
 - iAccuracy, [10](#)
- gpsPayload::altitude, [10](#)
 - fAltitude, [10](#)
 - iAltitude, [10](#)
- gpsPayload::bearing, [10](#)
 - fBearing, [11](#)
 - iBearing, [11](#)
- gpsPayload::latitude, [18](#)
 - dLat, [18](#)
 - iLat, [18](#)
- gpsPayload::longitude, [18](#)
 - dLong, [18](#)
 - iLong, [18](#)
- gpsPayload::speed, [22](#)
 - fSpeed, [22](#)
 - iSpeed, [22](#)
- Graph
 - Graph.h, [44](#)
- graph, [15](#)
 - root, [16](#)
- Graph.c
 - _XOPEN_SOURCE, [40](#)
 - edge_, [40](#)
 - Graph_addEdge, [40](#)
 - Graph_addNode, [40](#)
 - Graph_create, [41](#)
 - Graph_deleteEdge, [41](#)
 - Graph_deleteNode, [41](#)
 - Graph_disassemble, [41](#)
 - Graph_getEdgeWeight, [41](#)
 - Graph_getNeighbors, [42](#)
 - Graph_getNodes, [42](#)
 - Graph_isAdjacent, [42](#)
 - Graph_print, [43](#)
 - Graph_updateEdgeWeight, [43](#)
 - node_, [40](#)
- Graph.h
 - Graph, [44](#)

- Graph_addEdge, [44](#)
- Graph_addNode, [45](#)
- Graph_create, [45](#)
- Graph_deleteEdge, [45](#)
- Graph_deleteNode, [45](#)
- Graph_disassemble, [46](#)
- Graph_getEdgeWeight, [46](#)
- Graph_getNeighbors, [46](#)
- Graph_getNodes, [46](#)
- Graph_isAdjacent, [47](#)
- Graph_print, [47](#)
- Graph_updateEdgeWeight, [47](#)
- Graph_addEdge
 - adjlist.c, [35](#)
 - Graph.c, [40](#)
 - Graph.h, [44](#)
- Graph_addNode
 - adjlist.c, [35](#)
 - Graph.c, [40](#)
 - Graph.h, [45](#)
- Graph_create
 - adjlist.c, [36](#)
 - Graph.c, [41](#)
 - Graph.h, [45](#)
- Graph_deleteEdge
 - adjlist.c, [36](#)
 - Graph.c, [41](#)
 - Graph.h, [45](#)
- Graph_deleteNode
 - adjlist.c, [36](#)
 - Graph.c, [41](#)
 - Graph.h, [45](#)
- Graph_disassemble
 - adjlist.c, [36](#)
 - Graph.c, [41](#)
 - Graph.h, [46](#)
- Graph_getEdgeWeight
 - adjlist.c, [37](#)
 - Graph.c, [41](#)
 - Graph.h, [46](#)
- Graph_getNeighbors
 - adjlist.c, [37](#)
 - Graph.c, [42](#)
 - Graph.h, [46](#)
- Graph_getNodes
 - adjlist.c, [37](#)
 - Graph.c, [42](#)
 - Graph.h, [46](#)
- Graph_isAdjacent
 - adjlist.c, [37](#)
 - Graph.c, [42](#)
 - Graph.h, [47](#)
- Graph_print
 - adjlist.c, [38](#)
 - Graph.c, [43](#)
 - Graph.h, [47](#)
- Graph_updateEdgeWeight
 - adjlist.c, [38](#)
- Graph.c, [43](#)
- Graph.h, [47](#)
- hashtable.c
 - _XOPEN_SOURCE, [49](#)
 - hashtable_print, [49](#)
 - map_create, [49](#)
 - map_destroy, [49](#)
 - map_exists, [49](#)
 - map_insert, [49](#)
 - map_lookup, [50](#)
- hashtable_print
 - hashtable.c, [49](#)
 - map.c, [51](#)
- heap.c
 - heap_print, [31](#)
 - pqueue_create, [31](#)
 - pqueue_dequeue, [31](#)
 - pqueue_destroy, [31](#)
 - pqueue_enqueue, [31](#)
 - pqueue_reprioritize, [31](#)
 - pqueue_search, [31](#)
 - pqueue_size, [31](#)
- heap_print
 - heap.c, [31](#)
 - pqueue.c, [32](#)
- hexToDouble
 - zergProtos.c, [61](#)
 - zergProtos.h, [67](#)
- hexToInt
 - zergProtos.c, [61](#)
 - zergProtos.h, [67](#)
- hexToShort
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- hopLimit
 - ipv6Header, [17](#)
- iAccuracy
 - gpsPayload::accuracy, [10](#)
- iAltitude
 - gpsPayload::altitude, [10](#)
- iBearing
 - gpsPayload::bearing, [11](#)
- iLat
 - gpsPayload::latitude, [18](#)
- iLong
 - gpsPayload::longitude, [18](#)
- iParam2
 - cPayload::param2, [20](#)
- iSpeed
 - gpsPayload::speed, [22](#)
 - payload::sSpeed, [23](#)
- id
 - ipv4Header, [16](#)
 - ZergUnit, [25](#)
- ipHeaderLength
 - ipv4Header, [16](#)
- ipLength

- ipv4Header, 16
- ipv4Header, 16
 - checksum, 16
 - destIp, 16
 - dscp, 16
 - flags, 16
 - id, 16
 - ipHeaderLength, 16
 - ipLength, 16
 - protocol, 17
 - sourceIp, 17
 - ttl, 17
 - version, 17
 - zergStructs.h, 70
- ipv6Header, 17
 - destination, 17
 - flowLabel, 17
 - hopLimit, 17
 - nextHeader, 17
 - payloadLength, 17
 - source, 17
 - trafficClass, 17
 - version, 17
 - zergStructs.h, 70
- key
 - entry, 13
- latitude
 - gpsPayload, 15
- length
 - udpHeader, 23
- lengthOfData
 - pcapPacketHeader, 22
- linkLayerType
 - pcapFileHeader, 21
- loc
 - ZergUnit, 25
- longitude
 - gpsPayload, 15
- main
 - zergmap.c, 71
- majorVersion
 - pcapFileHeader, 21
- map
 - map.h, 53
- map.c
 - _XOPEN_SOURCE, 51
 - hashtable_print, 51
 - map_create, 51
 - map_destroy, 51
 - map_exists, 51
 - map_insert, 52
 - map_lookup, 52
- map.h
 - map, 53
 - map_create, 53
 - map_destroy, 53
 - map_exists, 54
 - map_insert, 54
 - map_lookup, 54
- map_create
 - hashtable.c, 49
 - map.c, 51
 - map.h, 53
- map_destroy
 - hashtable.c, 49
 - map.c, 51
 - map.h, 53
- map_exists
 - hashtable.c, 49
 - map.c, 51
 - map.h, 54
- map_insert
 - hashtable.c, 49
 - map.c, 52
 - map.h, 54
- map_lookup
 - hashtable.c, 50
 - map.c, 52
 - map.h, 54
- maxHitPoints
 - payload, 21
- maxLength
 - pcapFileHeader, 21
- microEpoch
 - pcapPacketHeader, 22
- minorVersion
 - pcapFileHeader, 21
- name
 - node_, 19
- next
 - edge_, 12
 - entry, 13
 - node_, 19
- nextHeader
 - ipv6Header, 17
- node_, 19
 - adjlist.c, 35
 - edges, 19
 - Graph.c, 40
 - name, 19
 - next, 19
- param1
 - cPayload, 12
- param2
 - cPayload, 12
- parseCapture
 - zergProtos.c, 62
 - zergProtos.h, 67
- payload, 20
 - armor, 21
 - currHitPoints, 21
 - maxHitPoints, 21
 - sSpeed, 21

- type, [21](#)
- zergStructs.h, [70](#)
- payload::sSpeed, [23](#)
 - fSpeed, [23](#)
 - iSpeed, [23](#)
- payloadLength
 - ipv6Header, [17](#)
- pcapFileHeader, [21](#)
 - accDelta, [21](#)
 - fileTypeId, [21](#)
 - gmtOffset, [21](#)
 - linkLayerType, [21](#)
 - majorVersion, [21](#)
 - maxLength, [21](#)
 - minorVersion, [21](#)
 - zergStructs.h, [70](#)
- pcapPacketHeader, [22](#)
 - fullLength, [22](#)
 - lengthOfData, [22](#)
 - microEpoch, [22](#)
 - unixEpoch, [22](#)
 - zergStructs.h, [71](#)
- pickPacketType
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- pqueue
 - pqueue.h, [33](#)
- pqueue.c
 - heap_print, [32](#)
 - pqueue_create, [32](#)
 - pqueue_dequeue, [32](#)
 - pqueue_destroy, [32](#)
 - pqueue_enqueue, [32](#)
 - pqueue_reprioritize, [32](#)
 - pqueue_search, [32](#)
 - pqueue_size, [32](#)
- pqueue.h
 - pqueue, [33](#)
 - pqueue_create, [33](#)
 - pqueue_dequeue, [33](#)
 - pqueue_destroy, [33](#)
 - pqueue_enqueue, [33](#)
 - pqueue_reprioritize, [33](#)
 - pqueue_search, [33](#)
 - pqueue_size, [33](#)
- pqueue_create
 - heap.c, [31](#)
 - pqueue.c, [32](#)
 - pqueue.h, [33](#)
- pqueue_dequeue
 - heap.c, [31](#)
 - pqueue.c, [32](#)
 - pqueue.h, [33](#)
- pqueue_destroy
 - heap.c, [31](#)
 - pqueue.c, [32](#)
 - pqueue.h, [33](#)
- pqueue_enqueue
 - heap.c, [31](#)
 - pqueue.c, [32](#)
 - pqueue.h, [33](#)
- pqueue_reprioritize
 - heap.c, [31](#)
 - pqueue.c, [32](#)
 - pqueue.h, [33](#)
- pqueue_search
 - heap.c, [31](#)
 - pqueue.c, [32](#)
 - pqueue.h, [33](#)
- pqueue_size
 - heap.c, [31](#)
 - pqueue.c, [32](#)
 - pqueue.h, [33](#)
- print_zergUnit
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- priority
 - _item, [7](#)
- protocol
 - ipv4Header, [17](#)
- README.md, [27](#)
- readCommand
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readEthernetPacket
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readGPS
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readIpv4Packet
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readIpv6Packet
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readMessage
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readPcapHeader
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readPcapPacket
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readStatus
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readUdpPacket
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readZerg
 - zergProtos.c, [62](#)
 - zergProtos.h, [67](#)
- readZergPacket
 - zergProtos.c, [62](#)

- zergProtos.h, 67
- root
 - graph, 16
- rotate3ByteInt
 - zergProtos.c, 62
 - zergProtos.h, 68
- rotateBack
 - zergProtos.c, 62
 - zergProtos.h, 68
- sSpeed
 - payload, 21
- seen
 - ZergUnit, 25
- sequenceId
 - zergPacket, 24
- size
 - _map, 8
 - _pqueue, 9
 - _vmap, 9
- source
 - ipv6Header, 17
- sourceId
 - zergPacket, 24
- sourceIp
 - ipv4Header, 17
- sourceMac
 - ethernetHeader, 14
- sourcePort
 - udpHeader, 23
- speed
 - gpsPayload, 15
- src/dijkstra/Dijkstra.c, 27
- src/dijkstra/Dijkstra.h, 28
- src/dpqueue/heap.c, 30
- src/dpqueue/pqueue.c, 31
- src/dpqueue/pqueue.h, 32
- src/graph/Graph.c, 38
- src/graph/Graph.h, 43
- src/graph/adjlist.c, 34
- src/map/hashtable.c, 48
- src/map/map.c, 50
- src/map/map.h, 52
- src/map/vmap.c, 55
- src/map/vmap.h, 57
- src/zerg/zergProtos.c, 59
- src/zerg/zergProtos.h, 64
- src/zerg/zergStructs.h, 69
- src/zergmap.c, 71
- status
 - ZergUnit, 25
- to
 - edge_, 12
- totalLength
 - zergPacket, 24
- trafficClass
 - ipv6Header, 17
- tTl
 - ipv4Header, 17
- type
 - payload, 21
 - zergPacket, 24
- udpHeader, 23
 - checksum, 23
 - destPort, 23
 - length, 23
 - sourcePort, 23
 - zergStructs.h, 71
- uiParam2
 - cPayload::param2, 20
- unixEpoch
 - pcapPacketHeader, 22
- validateHeader
 - zergProtos.c, 62
 - zergProtos.h, 68
- value
 - entry, 13
- version
 - ipv4Header, 17
 - ipv6Header, 17
 - zergPacket, 24
- vmap
 - vmap.h, 58
- vmap.c
 - _XOPEN_SOURCE, 56
 - vmap_create, 56
 - vmap_destroy, 56
 - vmap_exists, 56
 - vmap_insert, 56
 - vmap_lookup, 57
- vmap.h
 - vmap, 58
 - vmap_create, 58
 - vmap_destroy, 58
 - vmap_exists, 58
 - vmap_insert, 59
 - vmap_lookup, 59
- vmap_create
 - vmap.c, 56
 - vmap.h, 58
- vmap_destroy
 - vmap.c, 56
 - vmap.h, 58
- vmap_exists
 - vmap.c, 56
 - vmap.h, 58
- vmap_insert
 - vmap.c, 56
 - vmap.h, 59
- vmap_lookup
 - vmap.c, 57
 - vmap.h, 59
- weight
 - edge_, 12

- writeCommand
 - zergProtos.c, [62](#)
 - zergProtos.h, [68](#)
- writeEtherHeader
 - zergProtos.c, [62](#)
 - zergProtos.h, [68](#)
- writeGPS
 - zergProtos.c, [62](#)
 - zergProtos.h, [68](#)
- writelpv4Header
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- writeMessage
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- writePcapHeader
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- writePcapPacket
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- writeStatus
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- writeUdpHeader
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- writeZergHeader
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- Zerg_twoPaths
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- zergPacket, [24](#)
 - destinationId, [24](#)
 - sequenceId, [24](#)
 - sourceId, [24](#)
 - totalLength, [24](#)
 - type, [24](#)
 - version, [24](#)
 - zergStructs.h, [71](#)
- zergPayloadSize
 - zergProtos.c, [64](#)
- zergProtos.c
 - _GNU_SOURCE, [61](#)
 - checkEntry, [61](#)
 - create_unit, [61](#)
 - deleteRoute, [61](#)
 - fileCorruption, [61](#)
 - fscanNum, [63](#)
 - hexToDouble, [61](#)
 - hexToInt, [61](#)
 - hexToShort, [62](#)
 - parseCapture, [62](#)
 - pickPacketType, [62](#)
 - print_zergUnit, [62](#)
 - readCommand, [62](#)
 - readEthernetPacket, [62](#)
 - readGPS, [62](#)
 - readIpv4Packet, [62](#)
 - readIpv6Packet, [62](#)
 - readMessage, [62](#)
 - readPcapHeader, [62](#)
 - readPcapPacket, [62](#)
 - readStatus, [62](#)
 - readUdpPacket, [62](#)
 - readZerg, [62](#)
 - readZergPacket, [62](#)
 - rotate3ByteInt, [62](#)
 - rotateBack, [62](#)
 - validateHeader, [62](#)
 - writeCommand, [62](#)
 - writeEtherHeader, [62](#)
 - writeGPS, [62](#)
 - writelpv4Header, [63](#)
 - writeMessage, [63](#)
 - writePcapHeader, [63](#)
 - writePcapPacket, [63](#)
 - writeStatus, [63](#)
 - writeUdpHeader, [63](#)
 - writeZergHeader, [63](#)
 - Zerg_twoPaths, [63](#)
 - zergPayloadSize, [64](#)
 - zergUnit_distance, [63](#)
- zergProtos.h
 - checkEntry, [65](#)
 - create_unit, [65](#)
 - decimalDegreesToDMS, [65](#)
 - deleteRoute, [66](#)
 - fileCorruption, [67](#)
 - hexToDouble, [67](#)
 - hexToInt, [67](#)
 - hexToShort, [67](#)
 - parseCapture, [67](#)
 - pickPacketType, [67](#)
 - print_zergUnit, [67](#)
 - readCommand, [67](#)
 - readEthernetPacket, [67](#)
 - readGPS, [67](#)
 - readIpv4Packet, [67](#)
 - readIpv6Packet, [67](#)
 - readMessage, [67](#)
 - readPcapHeader, [67](#)
 - readPcapPacket, [67](#)
 - readStatus, [67](#)
 - readUdpPacket, [67](#)
 - readZerg, [67](#)
 - readZergPacket, [67](#)
 - rotate3ByteInt, [68](#)
 - rotateBack, [68](#)
 - validateHeader, [68](#)
 - writeCommand, [68](#)
 - writeEtherHeader, [68](#)
 - writeGPS, [68](#)
 - writelpv4Header, [68](#)
 - writeMessage, [68](#)

- writePcapHeader, [68](#)
- writePcapPacket, [68](#)
- writeStatus, [68](#)
- writeUdpHeader, [68](#)
- writeZergHeader, [68](#)
- Zerg_twoPaths, [68](#)
- zergUnit_distance, [68](#)
- zergStructs.h
 - cPayload, [70](#)
 - ethernetHeader, [70](#)
 - gpsPayload, [70](#)
 - ipv4Header, [70](#)
 - ipv6Header, [70](#)
 - payload, [70](#)
 - pcapFileHeader, [70](#)
 - pcapPacketHeader, [71](#)
 - udpHeader, [71](#)
 - zergPacket, [71](#)
 - ZergUnit, [71](#)
- ZergUnit, [24](#)
 - dupe, [25](#)
 - id, [25](#)
 - loc, [25](#)
 - seen, [25](#)
 - status, [25](#)
 - zergStructs.h, [71](#)
- zergUnit_distance
 - zergProtos.c, [63](#)
 - zergProtos.h, [68](#)
- zergmap.c
 - main, [71](#)