

# CITRIC

Generated by Doxygen 1.7.6.1

Thu Oct 24 2013 09:38:07



# Contents

<b>1</b>	<b>CITRIC introduction</b>	<b>1</b>
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	TAggregator Class Reference . . . . .	9
5.1.1	Detailed Description . . . . .	10
5.1.2	Member Function Documentation . . . . .	10
5.1.2.1	add . . . . .	10
5.1.2.2	dump . . . . .	10
5.1.3	Member Data Documentation . . . . .	10
5.1.3.1	Index . . . . .	10
5.1.3.2	Size . . . . .	10
5.1.3.3	WriteIndex . . . . .	10
5.2	TCauseAnalyzer Class Reference . . . . .	11
5.2.1	Detailed Description . . . . .	13
5.2.2	Constructor & Destructor Documentation . . . . .	13
5.2.2.1	TCauseAnalyzer . . . . .	13
5.2.3	Member Function Documentation . . . . .	13
5.2.3.1	add . . . . .	13

---

5.2.3.2	dump	14
5.2.4	Member Data Documentation	15
5.2.4.1	AlreadyOpenCauseCounter	15
5.2.4.2	BestDNSPlace	15
5.2.4.3	BestDNSTime	15
5.2.4.4	BestHTTPEventIndex	15
5.2.4.5	BestID	15
5.2.4.6	BestURLDelay	16
5.2.4.7	BestURLEventIndex	16
5.2.4.8	BestURLHash	16
5.2.4.9	BestUserEventIndex	16
5.2.4.10	CauseCallCounter	16
5.2.4.11	CauseCounter	16
5.2.4.12	DELDNSCauseCounter	16
5.2.4.13	DELSURLCauseCounter	16
5.2.4.14	DELURLCauseCounter	16
5.2.4.15	DNSCauseCounter	16
5.2.4.16	DNSCounter	16
5.2.4.17	DNSDelay	16
5.2.4.18	DNSNoNameFlowCounter	16
5.2.4.19	DNSUnknownCauseCounter	16
5.2.4.20	FoundDNSCount	16
5.2.4.21	FoundDNSIndex	16
5.2.4.22	HTTPCauseCounter	16
5.2.4.23	HTTPSCauseCounter	16
5.2.4.24	ID	16
5.2.4.25	IDLOverLengthCounter	16
5.2.4.26	IDLOverTokenCounter	17
5.2.4.27	LastUserTimeStamp	17
5.2.4.28	NoNameFlowCounter	17
5.2.4.29	ResolvedDNSUnknownCauseCounter	17
5.2.4.30	RPTDNSCauseCounter	17
5.2.4.31	SERVERCauseCounter	17
5.2.4.32	StatDNSTimes	17

---

5.2.4.33	StatDNSTimesCounter	17
5.2.4.34	StatURLTimes	17
5.2.4.35	StatURLTimesCounter	17
5.2.4.36	SURLCauseCounter	17
5.2.4.37	UnknownCauseCounter	17
5.2.4.38	URLCauseCounter	17
5.2.4.39	URLEventIndex	17
5.2.4.40	USERCauseCounter	17
5.2.4.41	UserDelay	17
5.2.4.42	UTreeCauseCounter	17
5.2.4.43	WhiteListCauseCounter	18
5.2.4.44	WorstDNSPlace	18
5.2.4.45	WorstDNSTime	18
5.3	TConnTrack Class Reference	18
5.3.1	Detailed Description	18
5.3.2	Constructor & Destructor Documentation	18
5.3.2.1	TConnTrack	18
5.3.3	Member Function Documentation	18
5.3.3.1	kill	18
5.3.3.2	kill	19
5.4	TDNS Class Reference	19
5.4.1	Detailed Description	20
5.4.2	Constructor & Destructor Documentation	20
5.4.2.1	TDNS	20
5.4.3	Member Function Documentation	21
5.4.3.1	clear	21
5.4.3.2	match	21
5.4.3.3	print	22
5.4.3.4	set	22
5.4.4	Member Data Documentation	23
5.4.4.1	CNAME	23
5.4.4.2	FlowIndex	23
5.4.4.3	IP	23
5.4.4.4	NAME	23

5.4.4.5	NextDNSIndex . . . . .	23
5.4.4.6	Resolved . . . . .	23
5.4.4.7	TimeStamp . . . . .	23
5.4.4.8	TTL . . . . .	24
5.5	TDNSHelper Class Reference . . . . .	24
5.5.1	Detailed Description . . . . .	26
5.5.2	Constructor & Destructor Documentation . . . . .	26
5.5.2.1	TDNSHelper . . . . .	26
5.5.3	Member Function Documentation . . . . .	26
5.5.3.1	add . . . . .	26
5.5.3.2	addToAnswerName . . . . .	27
5.5.3.3	addToQueryName . . . . .	28
5.5.3.4	clearAnswerName . . . . .	28
5.5.3.5	clearQueryName . . . . .	29
5.5.3.6	deleteRecord . . . . .	29
5.5.3.7	dump . . . . .	29
5.5.3.8	find . . . . .	30
5.5.3.9	getQueryName . . . . .	31
5.5.3.10	makeIPHash . . . . .	31
5.5.3.11	parseName . . . . .	32
5.5.4	Member Data Documentation . . . . .	32
5.5.4.1	AnswerName . . . . .	33
5.5.4.2	AnswerNameIndex . . . . .	33
5.5.4.3	DNSTable . . . . .	33
5.5.4.4	Length . . . . .	33
5.5.4.5	Packet . . . . .	33
5.5.4.6	QueryName . . . . .	33
5.5.4.7	QueryNameIndex . . . . .	33
5.6	TEventCollector Class Reference . . . . .	33
5.6.1	Detailed Description . . . . .	35
5.6.2	Constructor & Destructor Documentation . . . . .	35
5.6.2.1	TEventCollector . . . . .	36
5.6.3	Member Function Documentation . . . . .	36
5.6.3.1	addHTTPEvent . . . . .	36

5.6.3.2	<a href="#">addHTTPSEvent</a>	37
5.6.3.3	<a href="#">addRPTDNSEvent</a>	37
5.6.3.4	<a href="#">addURLEvent</a>	38
5.6.3.5	<a href="#">addUserEvent</a>	39
5.6.3.6	<a href="#">addUTreeEvent</a>	39
5.6.3.7	<a href="#">dump</a>	40
5.6.3.8	<a href="#">getLastUserEvent</a>	42
5.6.3.9	<a href="#">makeIPHash</a>	42
5.6.3.10	<a href="#">makeURLHash</a>	42
5.6.3.11	<a href="#">removeHTTPSEvent</a>	43
5.6.3.12	<a href="#">removeHTTPSEvent</a>	43
5.6.3.13	<a href="#">searchHTTPSEvent</a>	43
5.6.3.14	<a href="#">searchHTTPSEvent</a>	44
5.6.3.15	<a href="#">searchRPTDNSEvent</a>	44
5.6.3.16	<a href="#">searchSOFTURLEvent</a>	45
5.6.3.17	<a href="#">searchURLEvent</a>	46
5.6.3.18	<a href="#">searchUserEvent</a>	47
5.6.3.19	<a href="#">searchUTreeEvent</a>	47
5.6.4	<a href="#">Member Data Documentation</a>	48
5.6.4.1	<a href="#">AggregatedWindowTime</a>	48
5.6.4.2	<a href="#">Hash</a>	48
5.6.4.3	<a href="#">HTTPSEvent</a>	48
5.6.4.4	<a href="#">HTTPSEventCount</a>	48
5.6.4.5	<a href="#">HTTPSEventIndex</a>	48
5.6.4.6	<a href="#">HTTPSEvent</a>	48
5.6.4.7	<a href="#">HTTPSEventCount</a>	48
5.6.4.8	<a href="#">HTTPSEventIndex</a>	48
5.6.4.9	<a href="#">RPTDNSEvent</a>	48
5.6.4.10	<a href="#">RPTDNSEventIndex</a>	48
5.6.4.11	<a href="#">URLEvent</a>	49
5.6.4.12	<a href="#">URLEventIndex</a>	49
5.6.4.13	<a href="#">UserEvent</a>	49
5.6.4.14	<a href="#">UserEventCount</a>	49
5.6.4.15	<a href="#">UserEventIndex</a>	49

5.6.4.16	UTreeEvent	49
5.6.4.17	UTreeEventCount	49
5.6.4.18	UTreeEventIndex	49
5.6.4.19	WindowEndTime	49
5.7	TFlow Class Reference	49
5.7.1	Detailed Description	51
5.7.2	Constructor & Destructor Documentation	51
5.7.2.1	TFlow	51
5.7.3	Member Function Documentation	52
5.7.3.1	clear	52
5.7.3.2	getHash	52
5.7.3.3	match	52
5.7.3.4	print	53
5.7.4	Member Data Documentation	53
5.7.4.1	CausalTime	53
5.7.4.2	Cause	53
5.7.4.3	CauseReliability	54
5.7.4.4	Direction	54
5.7.4.5	DNSIndex	54
5.7.4.6	HTTPIndex	54
5.7.4.7	Identification	54
5.7.4.8	LocalIP	54
5.7.4.9	LocalPort	54
5.7.4.10	LocalSEQ	54
5.7.4.11	NextFlowIndex	54
5.7.4.12	NumberOfReceivedBytes	54
5.7.4.13	NumberOfReceivedPackets	55
5.7.4.14	NumberOfTransmittedBytes	55
5.7.4.15	NumberOfTransmittedPackets	55
5.7.4.16	ParentFlow	55
5.7.4.17	Protocol	55
5.7.4.18	RemoteIP	55
5.7.4.19	RemotePort	55
5.7.4.20	RemoteSEQ	55



5.7.4.21	Resolver	55
5.7.4.22	StartTime	55
5.7.4.23	Status	56
5.7.4.24	StopTime	56
5.7.4.25	TCPFlag	56
5.7.4.26	TreeIndex	56
5.8	TFlowAggregator Class Reference	56
5.8.1	Detailed Description	58
5.8.2	Constructor & Destructor Documentation	58
5.8.2.1	TFlowAggregator	58
5.8.3	Member Function Documentation	58
5.8.3.1	add	58
5.8.3.2	addFlow	59
5.8.3.3	addPacket	61
5.8.3.4	deleteFlow	61
5.8.3.5	dump	62
5.8.3.6	find	63
5.8.3.7	find	63
5.8.4	Member Data Documentation	64
5.8.4.1	AlreadyOpenCounter	64
5.8.4.2	Direction	64
5.8.4.3	EgressDNSFlowCounter	64
5.8.4.4	EgressFlowCounter	64
5.8.4.5	EgressHTTPFlowCounter	64
5.8.4.6	EgressHTTPSFlowCounter	64
5.8.4.7	EgressICMPFlowCounter	64
5.8.4.8	EgressOtherFlowCounter	64
5.8.4.9	EgressTCPFlowCounter	64
5.8.4.10	EgressUDPFlowCounter	64
5.8.4.11	IngressFlowCounter	64
5.8.4.12	IngressICMPFlowCounter	64
5.8.4.13	IngressOtherFlowCounter	64
5.8.4.14	IngressTCPFlowCounter	65
5.8.4.15	IngressUDPFlowCounter	65

5.8.4.16	InPacketCounter . . . . .	65
5.8.4.17	LocalIP . . . . .	65
5.8.4.18	NoLocalPacketCounter . . . . .	65
5.8.4.19	NonEmptyRRUDPCounter . . . . .	65
5.8.4.20	OutPacketCounter . . . . .	65
5.8.4.21	StartTime . . . . .	65
5.8.4.22	TotalPacketCounter . . . . .	65
5.9	TGZIP Class Reference . . . . .	65
5.9.1	Detailed Description . . . . .	66
5.9.2	Constructor & Destructor Documentation . . . . .	66
5.9.2.1	TGZIP . . . . .	66
5.9.3	Member Function Documentation . . . . .	66
5.9.3.1	free . . . . .	66
5.9.3.2	isInUse . . . . .	67
5.9.3.3	take . . . . .	67
5.9.3.4	uncompress . . . . .	67
5.9.4	Member Data Documentation . . . . .	68
5.9.4.1	InBuffer . . . . .	68
5.9.4.2	InLength . . . . .	68
5.9.4.3	InUse . . . . .	68
5.9.4.4	OutBuffer . . . . .	68
5.9.4.5	OutLength . . . . .	68
5.9.4.6	Processing . . . . .	68
5.9.4.7	strm . . . . .	68
5.10	THTTP Class Reference . . . . .	68
5.10.1	Detailed Description . . . . .	70
5.10.2	Constructor & Destructor Documentation . . . . .	70
5.10.2.1	THTTP . . . . .	70
5.10.3	Member Function Documentation . . . . .	70
5.10.3.1	clear . . . . .	70
5.10.3.2	print . . . . .	71
5.10.4	Member Data Documentation . . . . .	71
5.10.4.1	Chunked . . . . .	71
5.10.4.2	ContentType . . . . .	71

5.10.4.3	Encoding	71
5.10.4.4	FlowIndex	71
5.10.4.5	GZIPIndex	71
5.10.4.6	InByteCounter	72
5.10.4.7	LastHeaderTime	72
5.10.4.8	LastTailTime	72
5.10.4.9	OutByteCounter	72
5.10.4.10	ParseMicroState	72
5.10.4.11	ParseState	72
5.10.4.12	ParseSubState	72
5.10.4.13	PayloadSize	72
5.10.4.14	PostDotLength	72
5.10.4.15	RefBuffer	72
5.10.4.16	RefStat	72
5.10.4.17	Status	72
5.10.4.18	Time	73
5.10.4.19	TotalInByteCounter	73
5.10.4.20	TotalOutByteCounter	73
5.10.4.21	URLBuffer	73
5.11	THTTPEvent Class Reference	73
5.11.1	Constructor & Destructor Documentation	74
5.11.1.1	THTTPEvent	74
5.11.2	Member Function Documentation	74
5.11.2.1	clear	74
5.11.2.2	print	75
5.11.3	Member Data Documentation	75
5.11.3.1	CauseCount	75
5.11.3.2	DeltaT	75
5.11.3.3	EventCode	75
5.11.3.4	FlowIndex	75
5.11.3.5	TimeStamp	75
5.12	THTTPHelper Class Reference	76
5.12.1	Detailed Description	77
5.12.2	Constructor & Destructor Documentation	77

5.12.2.1	THTTPHelper	77
5.12.3	Member Function Documentation	77
5.12.3.1	add	77
5.12.3.2	checkURL	78
5.12.3.3	dump	78
5.12.3.4	getRef	79
5.12.3.5	parse	80
5.12.3.6	stripRef	81
5.12.4	Member Data Documentation	81
5.12.4.1	DeChunkedContent	82
5.12.4.2	GetRequestCounter	82
5.12.4.3	GZIP	82
5.12.4.4	GZIPIndex	82
5.12.4.5	RefCounter	82
5.12.4.6	SuccessRefCounter	82
5.13	THTTPSEvent Class Reference	82
5.13.1	Constructor & Destructor Documentation	83
5.13.1.1	THTTPSEvent	83
5.13.2	Member Function Documentation	83
5.13.2.1	clear	83
5.13.2.2	print	84
5.13.3	Member Data Documentation	84
5.13.3.1	CauseCount	84
5.13.3.2	DeltaT	84
5.13.3.3	EventCode	85
5.13.3.4	FlowIndex	85
5.13.3.5	TimeStamp	85
5.14	TLogger Class Reference	85
5.14.1	Detailed Description	86
5.14.2	Constructor & Destructor Documentation	86
5.14.2.1	TLogger	86
5.14.3	Member Function Documentation	86
5.14.3.1	initStatsLog	86
5.14.3.2	log	87

5.14.3.3	log	87
5.14.3.4	save	87
5.14.3.5	saveLog	88
5.14.3.6	saveStatsLog	88
5.14.4	Member Data Documentation	89
5.14.4.1	Buffer	89
5.14.4.2	FileName	89
5.14.4.3	LineCounter	89
5.14.4.4	Log	89
5.15	TPacketAnalyzer Class Reference	90
5.15.1	Detailed Description	91
5.15.2	Constructor & Destructor Documentation	91
5.15.2.1	TPacketAnalyzer	91
5.15.3	Member Function Documentation	91
5.15.3.1	dump	91
5.15.3.2	handleEvent	91
5.15.3.3	makePacket	93
5.15.4	Member Data Documentation	93
5.15.4.1	DestIP	93
5.15.4.2	DestPort	93
5.15.4.3	Identification	93
5.15.4.4	Length	93
5.15.4.5	PayloadIndex	93
5.15.4.6	Protocol	93
5.15.4.7	SourceIP	93
5.15.4.8	SourcePort	94
5.15.4.9	TCPACK	94
5.15.4.10	TCPFlag	94
5.15.4.11	TCPSEQ	94
5.15.4.12	TCPValid	94
5.15.4.13	Time	94
5.16	TPCAP Class Reference	94
5.16.1	Detailed Description	95
5.16.2	Constructor & Destructor Documentation	95

5.16.2.1	TPCAP	95
5.16.3	Member Function Documentation	95
5.16.3.1	applyFilter	95
5.16.3.2	close	96
5.16.3.3	getPacketEvent	96
5.16.3.4	openDevice	97
5.16.3.5	openFile	98
5.16.3.6	openNIC	98
5.16.4	Member Data Documentation	99
5.16.4.1	DeviceName	99
5.16.4.2	Filter	99
5.16.4.3	Handle	99
5.16.4.4	Length	99
5.16.4.5	Packet	99
5.16.4.6	TimeStamp	99
5.17	TRPTDNSEvent Class Reference	100
5.17.1	Constructor & Destructor Documentation	100
5.17.1.1	TRPTDNSEvent	100
5.17.2	Member Function Documentation	101
5.17.2.1	clear	101
5.17.2.2	print	101
5.17.3	Member Data Documentation	101
5.17.3.1	CauseCount	101
5.17.3.2	FlowIndex	102
5.17.3.3	Name	102
5.17.3.4	TimeStamp	102
5.18	TSettings Class Reference	102
5.18.1	Detailed Description	102
5.19	TSettings Class Reference	102
5.19.1	Constructor & Destructor Documentation	103
5.19.1.1	TSettings	103
5.19.2	Member Function Documentation	103
5.19.2.1	dump	103
5.19.2.2	parseFile	104

5.19.2.3	testWhiteList	105
5.19.2.4	testWhiteList	105
5.19.3	Member Data Documentation	105
5.19.3.1	DefinedSettings	105
5.19.3.2	DefinedValues	105
5.19.3.3	Name	105
5.19.3.4	WhiteIPHigh	105
5.19.3.5	WhiteIPLow	105
5.19.3.6	WhiteIPTotal	105
5.19.3.7	WhiteName	105
5.19.3.8	WhiteNameTotal	105
5.20	TTree Class Reference	106
5.20.1	Detailed Description	106
5.20.2	Constructor & Destructor Documentation	107
5.20.2.1	TTree	107
5.20.3	Member Function Documentation	107
5.20.3.1	clear	107
5.20.3.2	print	107
5.20.4	Member Data Documentation	108
5.20.4.1	DNSOther	108
5.20.4.2	ID	108
5.20.4.3	MaxDepth	108
5.20.4.4	NumberOfFlows	108
5.20.4.5	RootCause	108
5.20.4.6	RootFlow	108
5.20.4.7	StartTime	108
5.20.4.8	StopTime	109
5.21	TUDPUA Class Reference	109
5.21.1	Detailed Description	110
5.21.2	Constructor & Destructor Documentation	110
5.21.2.1	TUDPUA	110
5.21.3	Member Function Documentation	110
5.21.3.1	close	110
5.21.3.2	dump	110

5.21.3.3	<a href="#">getEvent</a>	110
5.21.3.4	<a href="#">open</a>	110
5.21.3.5	<a href="#">processEvent</a>	111
5.21.4	<a href="#">Member Data Documentation</a>	111
5.21.4.1	<a href="#">Event</a>	111
5.21.4.2	<a href="#">EventCode</a>	111
5.21.4.3	<a href="#">Message</a>	111
5.21.4.4	<a href="#">Process</a>	112
5.21.4.5	<a href="#">SocketDescriptor</a>	112
5.21.4.6	<a href="#">TimeStamp</a>	112
5.22	<a href="#">TURLEvent Class Reference</a>	112
5.22.1	<a href="#">Constructor &amp; Destructor Documentation</a>	113
5.22.1.1	<a href="#">TURLEvent</a>	113
5.22.2	<a href="#">Member Function Documentation</a>	113
5.22.2.1	<a href="#">clear</a>	113
5.22.2.2	<a href="#">print</a>	113
5.22.3	<a href="#">Member Data Documentation</a>	114
5.22.3.1	<a href="#">CauseCount</a>	114
5.22.3.2	<a href="#">FlowIndex</a>	114
5.22.3.3	<a href="#">SUBURL</a>	114
5.22.3.4	<a href="#">TimeStamp</a>	114
5.22.3.5	<a href="#">URL</a>	114
5.23	<a href="#">TUserEvent Class Reference</a>	114
5.23.1	<a href="#">Constructor &amp; Destructor Documentation</a>	115
5.23.1.1	<a href="#">TUserEvent</a>	115
5.23.2	<a href="#">Member Function Documentation</a>	115
5.23.2.1	<a href="#">clear</a>	115
5.23.2.2	<a href="#">print</a>	116
5.23.3	<a href="#">Member Data Documentation</a>	116
5.23.3.1	<a href="#">CauseCount</a>	116
5.23.3.2	<a href="#">EventCode</a>	116
5.23.3.3	<a href="#">Process</a>	117
5.23.3.4	<a href="#">TimeStamp</a>	117
5.24	<a href="#">TUTreeEvent Class Reference</a>	117



5.24.1	Constructor & Destructor Documentation . . . . .	117
5.24.1.1	TUTreeEvent . . . . .	117
5.24.2	Member Function Documentation . . . . .	118
5.24.2.1	clear . . . . .	118
5.24.2.2	print . . . . .	118
5.24.3	Member Data Documentation . . . . .	119
5.24.3.1	CauseCount . . . . .	119
5.24.3.2	DeltaT . . . . .	119
5.24.3.3	TimeStamp . . . . .	119
5.24.3.4	TreeIndex . . . . .	119
<b>6</b>	<b>File Documentation</b>	<b>121</b>
6.1	src/Aggregator.h File Reference . . . . .	121
6.1.1	Detailed Description . . . . .	121
6.2	src/CauseAnalyzer.cpp File Reference . . . . .	122
6.2.1	Detailed Description . . . . .	123
6.2.2	Variable Documentation . . . . .	123
6.2.2.1	ConnTrack . . . . .	123
6.2.2.2	DNS . . . . .	123
6.2.2.3	DNSHelper . . . . .	123
6.2.2.4	ENDPROG . . . . .	123
6.2.2.5	EventCollector . . . . .	123
6.2.2.6	Flow . . . . .	123
6.2.2.7	FlowAggregator . . . . .	123
6.2.2.8	HTTP . . . . .	123
6.2.2.9	HTTPHelper . . . . .	123
6.2.2.10	Logger . . . . .	123
6.2.2.11	PacketAnalyzer . . . . .	123
6.2.2.12	PCAP . . . . .	123
6.2.2.13	Settings . . . . .	124
6.2.2.14	Tree . . . . .	124
6.3	src/CauseAnalyzer.h File Reference . . . . .	124
6.3.1	Detailed Description . . . . .	125
6.3.2	Define Documentation . . . . .	125

6.3.2.1	TREE_BUFFER_SIZE	125
6.4	src/CITRIC.cpp File Reference	125
6.4.1	Detailed Description	126
6.4.2	Function Documentation	127
6.4.2.1	main	127
6.4.2.2	sigproc	127
6.4.3	Variable Documentation	129
6.4.3.1	CauseAnalyzer	129
6.4.3.2	ConnTrack	129
6.4.3.3	DNSHelper	129
6.4.3.4	DumpMessage	129
6.4.3.5	ENDPROG	129
6.4.3.6	EventCollector	129
6.4.3.7	FlowAggregator	129
6.4.3.8	HTTPHelper	129
6.4.3.9	Logger	129
6.4.3.10	PacketAnalyzer	129
6.4.3.11	PCAP	129
6.4.3.12	Settings	129
6.4.3.13	UDPUA	129
6.5	src/CITRIC.h File Reference	129
6.5.1	Detailed Description	130
6.5.2	Function Documentation	130
6.5.2.1	sigproc	130
6.6	src/ConnTrack.cpp File Reference	132
6.6.1	Detailed Description	132
6.6.2	Variable Documentation	133
6.6.2.1	Flow	133
6.6.2.2	Settings	133
6.7	src/ConnTrack.h File Reference	133
6.7.1	Detailed Description	135
6.8	src/DNS.cpp File Reference	135
6.8.1	Detailed Description	136
6.9	src/DNS.h File Reference	136

6.9.1	Detailed Description	137
6.9.2	Define Documentation	138
6.9.2.1	MAXDOMAINNAMELENGTH	138
6.10	src/DNSHelper.cpp File Reference	138
6.10.1	Detailed Description	138
6.10.2	Variable Documentation	139
6.10.2.1	CauseAnalyzer	139
6.10.2.2	DNS	139
6.10.2.3	EventCollector	139
6.10.2.4	Flow	139
6.10.2.5	FlowAggregator	139
6.10.2.6	Logger	139
6.10.2.7	PacketAnalyzer	139
6.10.2.8	PCAP	139
6.11	src/DNSHelper.h File Reference	139
6.11.1	Detailed Description	141
6.11.2	Define Documentation	141
6.11.2.1	DNSBUFFERSIZE	141
6.11.2.2	DNSHASHSIZE	141
6.12	src/EventCollector.cpp File Reference	141
6.12.1	Detailed Description	142
6.12.2	Variable Documentation	143
6.12.2.1	DNS	143
6.12.2.2	Flow	143
6.12.2.3	FlowAggregator	143
6.12.2.4	HTTP	143
6.12.2.5	Logger	143
6.12.2.6	PacketAnalyzer	143
6.12.2.7	Settings	143
6.13	src/EventCollector.h File Reference	143
6.13.1	Detailed Description	144
6.13.2	Define Documentation	145
6.13.2.1	CAUSE_HTTPDATAPUSH	145
6.13.2.2	CAUSE_HTTPDATA RECEIVED	145

6.13.2.3	CAUSE_HTTPSIZEDECREASE	145
6.13.2.4	CAUSE_HTTPWITHDRAWN	145
6.13.2.5	EVENTBUFFERSIZE	145
6.13.2.6	RPTDNSEVENTBUFFERSIZE	145
6.13.2.7	URLEVENTBUFFERSIZE	145
6.13.2.8	URLEVENTHASHSIZE	145
6.14	src/Flow.cpp File Reference	145
6.14.1	Detailed Description	146
6.14.2	Variable Documentation	146
6.14.2.1	DNS	146
6.15	src/Flow.h File Reference	146
6.15.1	Detailed Description	148
6.15.2	Define Documentation	148
6.15.2.1	CAUSE_ALREADYOPEN	148
6.15.2.2	CAUSE_DNS	148
6.15.2.3	CAUSE_DNS_REPEAT	148
6.15.2.4	CAUSE_HTTP_GEN	148
6.15.2.5	CAUSE_HTTP_REFERERER	148
6.15.2.6	CAUSE_HTTP_SOFTURL	148
6.15.2.7	CAUSE_HTTP_URL	148
6.15.2.8	CAUSE_HTTPS_GEN	148
6.15.2.9	CAUSE_PROTODNS	149
6.15.2.10	CAUSE_SERVER	149
6.15.2.11	CAUSE_UNKNOWN	149
6.15.2.12	CAUSE_USER	149
6.15.2.13	CAUSE_UTREE	149
6.15.2.14	CAUSE_WHITELIST	149
6.15.2.15	CAUSEQ_ID_NOTIME	149
6.15.2.16	CAUSEQ_ID_SOFTTIME	149
6.15.2.17	CAUSEQ_ID_TIME	149
6.15.2.18	CAUSEQ_NOID_NOTIME	149
6.15.2.19	CAUSEQ_NOID_SOFTTIME	149
6.15.2.20	CAUSEQ_NOID_TIME	149
6.15.2.21	CAUSEQ_SOFTID_NOTIME	149

6.15.2.22 CAUSEQ_SOFTID_SOFTTIME . . . . .	149
6.15.2.23 CAUSEQ_SOFTID_TIME . . . . .	149
6.15.2.24 CAUSEQ_TIME . . . . .	149
6.15.2.25 EGRESS . . . . .	149
6.15.2.26 FLOWHASHSIZE . . . . .	149
6.15.2.27 INGRESS . . . . .	149
6.15.2.28 RESOLVER_DNSNAME . . . . .	149
6.15.2.29 RESOLVER_NONAME . . . . .	149
6.15.2.30 RESOLVER_OTHERNAME . . . . .	149
6.15.2.31 RESOLVER_PROTO_DNS . . . . .	149
6.15.2.32 RESOLVER_PROTO_OTHER . . . . .	149
6.15.2.33 RESOLVER_PROTO_STATIC . . . . .	150
6.15.2.34 RESOLVER_STATICNAME . . . . .	150
6.15.2.35 RESOLVER_UNDEFINED . . . . .	150
6.15.2.36 UNDEFINED_DIR . . . . .	150
6.16 src/FlowAggregator.cpp File Reference . . . . .	150
6.16.1 Detailed Description . . . . .	150
6.16.2 Variable Documentation . . . . .	151
6.16.2.1 CauseAnalyzer . . . . .	151
6.16.2.2 DNSHelper . . . . .	151
6.16.2.3 Flow . . . . .	151
6.16.2.4 FlowTable . . . . .	151
6.16.2.5 HTTPHelper . . . . .	151
6.16.2.6 Logger . . . . .	151
6.16.2.7 PacketAnalyzer . . . . .	151
6.16.2.8 PCAP . . . . .	151
6.16.2.9 UDPUA . . . . .	151
6.17 src/FlowAggregator.h File Reference . . . . .	151
6.17.1 Detailed Description . . . . .	152
6.17.2 Define Documentation . . . . .	153
6.17.2.1 FLOWBUFFERSIZE . . . . .	153
6.18 src/GZIP.cpp File Reference . . . . .	153
6.18.1 Detailed Description . . . . .	154
6.18.2 Define Documentation . . . . .	154

6.18.2.1	ENABLE_ZLIB_GZIP	154
6.18.2.2	windowBits	154
6.18.3	Variable Documentation	154
6.18.3.1	PacketAnalyzer	154
6.19	src/GZIP.h File Reference	154
6.19.1	Detailed Description	155
6.19.2	Define Documentation	156
6.19.2.1	GZIP_BUFFERSIZE	156
6.20	src/HFSM.h File Reference	156
6.20.1	Detailed Description	157
6.20.2	Define Documentation	157
6.20.2.1	HFSM1_BODY	157
6.20.2.2	HFSM1_CHUNK	157
6.20.2.3	HFSM1_HEADER	157
6.20.2.4	HFSM1_IDLE	158
6.20.2.5	HFSM1_NO_HTTP	158
6.20.2.6	HFSM1_PARSED	158
6.20.2.7	HFSM2_	158
6.20.2.8	HFSM2_CHAR_RECEIVED	158
6.20.2.9	HFSM2_CHUNKED	158
6.20.2.10	HFSM2_CONTENT	158
6.20.2.11	HFSM2_CONTENTLENGTH	158
6.20.2.12	HFSM2_CONTENTTYPE	158
6.20.2.13	HFSM2_DOT_RECEIVED	158
6.20.2.14	HFSM2_ENCODING	158
6.20.2.15	HFSM2_IDLE	158
6.20.2.16	HFSM2_MOVED	158
6.20.2.17	HFSM2_NEWLINERECEIVED	158
6.20.2.18	HFSM2_OVERFLOW	158
6.20.2.19	HFSM2_WHITELINE	158
6.21	src/HTTP.cpp File Reference	158
6.21.1	Detailed Description	159
6.21.2	Variable Documentation	160
6.21.2.1	Flow	160

6.21.2.2	Tree	160
6.22	src/HTTP.h File Reference	160
6.22.1	Detailed Description	161
6.22.2	Define Documentation	161
6.22.2.1	HTTPSTATUS_RECEIVED	161
6.22.2.2	HTTPSTATUS_SENT	161
6.22.2.3	HTTPSTATUS_UNDEFINED	161
6.22.2.4	HTTPSTATUS_WAITINGFORSEND	162
6.22.2.5	REFSTAT_GETSEEN	162
6.22.2.6	REFSTAT_NOREF	162
6.22.2.7	REFSTAT_REF	162
6.22.2.8	REFSTAT_UNDEFINED	162
6.23	src/HTTPHelper.cpp File Reference	162
6.23.1	Detailed Description	163
6.23.2	Variable Documentation	163
6.23.2.1	EventCollector	163
6.23.2.2	Flow	163
6.23.2.3	FlowAggregator	163
6.23.2.4	HTTP	163
6.23.2.5	Logger	163
6.23.2.6	PacketAnalyzer	163
6.23.2.7	PatternContentTypeFlash	163
6.23.2.8	PatternContentTypeJava	163
6.23.2.9	PatternContentTypeJS	164
6.23.2.10	PatternContentTypeJSON	164
6.23.2.11	PatternContentTypeText	164
6.23.2.12	PatternEncodingGZIP	164
6.23.2.13	PatternHTTP	164
6.23.2.14	Patternhttp	164
6.23.2.15	PatternLocation	164
6.23.2.16	PatternTransfer	164
6.23.2.17	PatternWhiteLine	164
6.23.2.18	PCAP	164
6.23.2.19	TLD	164

6.24	src/HTTPHelper.h File Reference	165
6.24.1	Detailed Description	166
6.24.2	Define Documentation	166
6.24.2.1	BINARYCONTENT	166
6.24.2.2	HTTP_BUFFER_SIZE	166
6.24.2.3	TOTAL_GZIP	166
6.25	src/Logger.cpp File Reference	166
6.25.1	Detailed Description	167
6.25.2	Variable Documentation	168
6.25.2.1	CauseDesc	168
6.25.2.2	EventCollector	168
6.25.2.3	Facility	168
6.25.2.4	Flow	168
6.25.2.5	FlowAggregator	168
6.25.2.6	HTTP	168
6.25.2.7	PacketAnalyzer	168
6.25.2.8	PCAP	168
6.25.2.9	Protocol	168
6.25.2.10	Severity	168
6.26	src/Logger.h File Reference	168
6.26.1	Detailed Description	170
6.26.2	Define Documentation	170
6.26.2.1	ALERT	170
6.26.2.2	CRITICAL	170
6.26.2.3	DEBUG	170
6.26.2.4	EMERGENCY	170
6.26.2.5	ERROR	170
6.26.2.6	FAC_CAUSE	171
6.26.2.7	FAC_DNS	171
6.26.2.8	FAC_EVENT	171
6.26.2.9	FAC_FLOW	171
6.26.2.10	FAC_HTTP	171
6.26.2.11	FAC_MAIN	171
6.26.2.12	FAC_PACKET	171



6.26.2.13	INFORMATIONAL	171
6.26.2.14	LOGSIZE	171
6.26.2.15	NOTICE	171
6.26.2.16	WARNING	171
6.27	src/PacketAnalyzer.cpp File Reference	171
6.27.1	Detailed Description	172
6.27.2	Variable Documentation	172
6.27.2.1	FlowAggregator	172
6.27.2.2	PCAP	172
6.27.2.3	Settings	172
6.28	src/PacketAnalyzer.h File Reference	172
6.28.1	Detailed Description	173
6.29	src/PCAP.cpp File Reference	174
6.29.1	Detailed Description	174
6.29.2	Variable Documentation	175
6.29.2.1	Logger	175
6.30	src/PCAP.h File Reference	175
6.30.1	Detailed Description	176
6.31	src/pdu.h File Reference	176
6.31.1	Detailed Description	177
6.31.2	Define Documentation	178
6.31.2.1	DEST_PORT_OFFSET	178
6.31.2.2	DNS_IDENTIFICATION	178
6.31.2.3	DNS_NUMBER_OF_ANSWERS_OFFSET	178
6.31.2.4	DNS_NUMBER_OF_QUESTIONS_OFFSET	178
6.31.2.5	DNS_OPCODE_OFFSET	178
6.31.2.6	DNS_QUESTIONS_OFFSET	178
6.31.2.7	DNS_RCODE_OFFSET	178
6.31.2.8	ETH_PROT_OFFSET	178
6.31.2.9	ETHERNET_LENGTH	178
6.31.2.10	ICMP	178
6.31.2.11	ICMP_CODE_OFFSET	178
6.31.2.12	ICMP_ID_OFFSET	178
6.31.2.13	ICMP_LENGTH	178

6.31.2.14 ICMP_SEQ_OFFSET . . . . .	178
6.31.2.15 ICMP_TYPE_OFFSET . . . . .	178
6.31.2.16 IP_DEST_OFFSET . . . . .	178
6.31.2.17 IP_LENGTH_OFFSET . . . . .	178
6.31.2.18 IP_MIN_LENGTH . . . . .	178
6.31.2.19 IP_PROT_OFFSET . . . . .	178
6.31.2.20 IP_SOURCE_OFFSET . . . . .	179
6.31.2.21 IP_VERSION_OFFSET . . . . .	179
6.31.2.22 OTHER . . . . .	179
6.31.2.23 SOURCE_PORT_OFFSET . . . . .	179
6.31.2.24 TCP . . . . .	179
6.31.2.25 TCP_ACK_OFFSET . . . . .	179
6.31.2.26 TCP_FLAG_OFFSET . . . . .	179
6.31.2.27 TCP_LENGTH_OFFSET . . . . .	179
6.31.2.28 TCP_SEQ_OFFSET . . . . .	179
6.31.2.29 UDP . . . . .	179
6.31.2.30 UDP_LENGTH . . . . .	179
6.31.2.31 UDP_LENGTH . . . . .	179
6.32 src/Settings.cpp File Reference . . . . .	179
6.32.1 Detailed Description . . . . .	180
6.32.2 Variable Documentation . . . . .	181
6.32.2.1 Logger . . . . .	181
6.33 src/Settings.h File Reference . . . . .	181
6.33.1 Detailed Description . . . . .	182
6.33.2 Define Documentation . . . . .	182
6.33.2.1 DELTA_T_DNS . . . . .	182
6.33.2.2 DELTA_T_DNS_DEL . . . . .	182
6.33.2.3 DELTA_T_DNS_RPT . . . . .	182
6.33.2.4 DELTA_T_HTTP . . . . .	182
6.33.2.5 DELTA_T_HTTPS . . . . .	182
6.33.2.6 DELTA_T_URL . . . . .	182
6.33.2.7 DELTA_T_URL_DEL . . . . .	182
6.33.2.8 DELTA_T_USER . . . . .	183
6.33.2.9 DELTA_T_UTREE . . . . .	183

6.33.2.10 DNS_PORTPATCH . . . . .	183
6.33.2.11 IDL_MAX_LENGTH . . . . .	183
6.33.2.12 IDL_MAX_TOKENS . . . . .	183
6.33.2.13 IPS_ENABLE . . . . .	183
6.33.2.14 SETTINGSSIZE . . . . .	183
6.34 src/Tree.cpp File Reference . . . . .	183
6.34.1 Detailed Description . . . . .	184
6.34.2 Variable Documentation . . . . .	184
6.34.2.1 Flow . . . . .	184
6.35 src/Tree.h File Reference . . . . .	184
6.35.1 Detailed Description . . . . .	185
6.36 src/UDPUA.cpp File Reference . . . . .	185
6.36.1 Detailed Description . . . . .	186
6.36.2 Variable Documentation . . . . .	186
6.36.2.1 EventCollector . . . . .	186
6.36.2.2 PacketAnalyzer . . . . .	187
6.36.2.3 PCAP . . . . .	187
6.37 src/UDPUA.h File Reference . . . . .	187
6.37.1 Detailed Description . . . . .	188
6.37.2 Define Documentation . . . . .	188
6.37.2.1 UDP_Port . . . . .	188



# Chapter 1

## CITRIC introduction

CITRIC is an experimental causal detector that analyzes PCAP-data from a device. It aggregates packets in bidirectional flows and it organizes the flows in causal trees.

A causal tree is a group of flows with a causal relationship. Causal relationship is determined by time-intervals and in some cases content.

To start CITRIC: `sudo CITRIC x.x.x.x y`

with `x.x.x.x` is the address of the observed computer

and `y` is the name of the observed interface.

If `y` ends with a number it is assumed to be a device e.g. `eth0`.

If `y` ends with something else it is assumed to be a file in PCAP-format.

In both cases CITRIC is terminated with a Control-C sequence.

This is an experimental version for research.

### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

### Author

Pieter Burghouwt

### Version

Revision 2.0

### Date

Tuesday, May 7, 2013



## Chapter 2

# Class Index

### 2.1 Class Hierarchy

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

TAggregator . . . . .	9
TCauseAnalyzer . . . . .	11
TDNSHelper . . . . .	24
TFlowAggregator . . . . .	56
THTTPHelper . . . . .	76
TConnTrack . . . . .	18
TDNS . . . . .	19
TEventCollector . . . . .	33
TFlow . . . . .	49
TGZIP . . . . .	65
THTTP . . . . .	68
THTTPEvent . . . . .	73
THTTPSEvent . . . . .	82
TLogger . . . . .	85
TPacketAnalyzer . . . . .	90
TPCAP . . . . .	94
TRPTDNSEvent . . . . .	100
TSettinggs . . . . .	102
TSettings . . . . .	102
TTree . . . . .	106
TUDPUA . . . . .	109
TURLEvent . . . . .	112
TUserEvent . . . . .	114
TUTreeEvent . . . . .	117





## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">TAggregator</a>	Baseclass for aggregators. An aggregator is an object that aggregates multiple data in a combined datastructure or element of a small size. Examples of aggregating elements are Flows, Tree, and DNS-records . . . . .	9
<a href="#">TCauseAnalyzer</a>	Class that manages causal trees and sorts new flows in it . . . . .	11
<a href="#">TConnTrack</a>	Class that can kill an established connection . . . . .	18
<a href="#">TDNS</a>	Storage class for DNS-data. Objects of this class are managed by DNSHelper . . . . .	19
<a href="#">TDNSHelper</a>	Class specialized in aggregating DNS events. It only processes replies from egress UDP DNS-flows. DNS objects are stored as a chained hashtable of unidirectional linked lists. The hashtable is based on a modulo 256 sum of the IP-address. This results in a fast lookup by IP-address. Newer records with the same name and IP-address are replaced. TODO: cleanup-routine that checks one record at a time on expiry . . . . .	24
<a href="#">TEventCollector</a>	Class specialized in aggregating events that can cause new traffic. Every potential event is buffered here. Hashing is used for fast lookup. FIFO's are used to prevent overflow . . . . .	33
<a href="#">TFlow</a>	Storage class for a bidirectional flow . . . . .	49
<a href="#">TFlowAggregator</a>	Important class that manages the flows . . . . .	56

<a href="#">TGZIP</a>	Class for deflating gzip data. Wraps around zlib . . . . .	65
<a href="#">THTTP</a>	Storage class for HTTP-events. Every HTTP-flow or HTTPS-flow that can potentially trigger new traffi by embedded URL's has an object of this class . . . . .	68
<a href="#">THTTPEvent</a>	. . . . .	73
<a href="#">THTTPHelper</a>	Class that aggregates HTTP traffic. It aggregates information that predicts the cause of new traffic flows <a href="#">THTTPHelper</a> creates a static array of HTTP_BUFFER_SIZE (65536) HTTP-objects. The Cluster-Aggregator uses this class to check for potential HTTP-parent flows. <a href="#">TFlowAggregator</a> updates the information of this class . . . . .	76
<a href="#">THTTPSEvent</a>	. . . . .	82
<a href="#">TLogger</a>	Class that aggregates log messages in flat file format in RAM . . . .	85
<a href="#">TPacketAnalyzer</a>	Class for analyzing an arrived packet . . . . .	90
<a href="#">TPCAP</a>	The "singleton" object of <a href="#">TPCAP</a> is a wrapper around the libpcap library. It can read from a device or a file and return a pointer to a received packet . . . . .	94
<a href="#">TRPTDNSEvent</a>	. . . . .	100
<a href="#">TSettings</a>	Class for parsing and distributing the settings of the application . . .	102
<a href="#">TSettings</a>	. . . . .	102
<a href="#">TTree</a>	Storage class for a tree of network flows with a causal relationship .	106
<a href="#">TUDPUA</a>	One object of this class (class should be used as singleton) recieves and filters user events, received from UDP with a special agent. - Received results can be polled in a Event-loop . . . . .	109
<a href="#">TURLEvent</a>	. . . . .	112
<a href="#">TUserEvent</a>	. . . . .	114
<a href="#">TUTreeEvent</a>	. . . . .	117

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">Aggregator.h</a>	This file contains the base class <a href="#">TAggregator</a> . . . . .	121
src/ <a href="#">CauseAnalyzer.cpp</a>	This file contains the operators of the <a href="#">TCauseAnalyzer</a> class . . . . .	122
src/ <a href="#">CauseAnalyzer.h</a>	This file contains the prototypes of the <a href="#">TCauseAnalyzer</a> class . . . . .	124
src/ <a href="#">CITRIC.cpp</a>	This file contains the main file of CITRIC. CITRIC is an experimental causal detector that analyzes PCAP-data from a device. It aggregates packets in bidirectional flows and it organizes the flows in causal trees. A causal tree is a group of flows with a causal relationship. Causal relationship is determined by time-intervals and in some cases content. This is an experimental version for research . . .	125
src/ <a href="#">CITRIC.h</a>	This file contains the prototypes of the CITRIC class . . . . .	129
src/ <a href="#">ConnTrack.cpp</a>	This file contains the operators of the <a href="#">TConnTrack</a> class . . . . .	132
src/ <a href="#">ConnTrack.h</a>	This file contains the prototypes of the <a href="#">TConnTrack</a> class . . . . .	133
src/ <a href="#">DNS.cpp</a>	This file contains the operators of the <a href="#">TDNS</a> class . . . . .	135
src/ <a href="#">DNS.h</a>	This file contains the prototypes of the <a href="#">TDNS</a> class . . . . .	136
src/ <a href="#">DNSHelper.cpp</a>	This file contains the operators of the <a href="#">TDNSHelper</a> class . . . . .	138
src/ <a href="#">DNSHelper.h</a>	This file contains the prototypes of the <a href="#">TDNSHelper</a> class . . . . .	139
src/ <a href="#">EventCollector.cpp</a>	This file contains the operators of the EventCollector class . . . . .	141

src/EventCollector.h	
This file contains the prototypes of the TEventCollector class	143
src/Flow.cpp	
This file contains the operators of the TFlow class	145
src/Flow.h	
This file contains the prototypes of the TFlow class	146
src/FlowAggregator.cpp	
This file contains the operators of the TFlowAggregator class	150
src/FlowAggregator.h	
This file contains the prototypes of the TFlowAggregator class	151
src/GZIP.cpp	
This file contains the operators of the TGZIP class	153
src/GZIP.h	
This file contains the prototypes of the TGZIP class	154
src/HFSM.h	
This file contains HFSM-state definitions	156
src/HTTP.cpp	
This file contains the operators of the THTTP class	158
src/HTTP.h	
This file contains the prototypes of the THTTP class	160
src/HTTPHelper.cpp	
This file contains the operators of the THTTPAggregator class	162
src/HTTPHelper.h	
This file contains the prototypes of the THTTPHelper class	165
src/Logger.cpp	
This file contains the operators of the TLogger class	166
src/Logger.h	
This file contains the prototypes of the TLogger class	168
src/PacketAnalyzer.cpp	
This file contains the operators of the TPacketAnalyzer class	171
src/PacketAnalyzer.h	
This file contains the prototypes of the TPacketAnalyzer class	172
src/PCAP.cpp	
This file contains the operators of the TPCAP class	174
src/PCAP.h	
This file contains the prototypes of the TPCAP class	175
src/pdu.h	
This file contains PDU-fields, places and values	176
src/Settings.cpp	
This file contains the operators of the TSettings class	179
src/Settings.h	
This file contains the prototypes of the TSettings class	181
src/Tree.cpp	
This file contains the operators of the TTree class	183
src/Tree.h	
This file contains the prototypes of the TTree class	184
src/UDPUA.cpp	
This file contains the operators of the TUDPUA class	185
src/UDPUA.h	
This file contains the prototypes of the UDPUA class	187

## Chapter 5

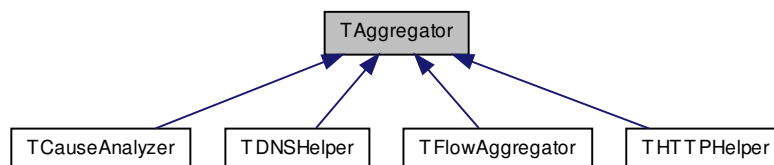
# Class Documentation

### 5.1 TAggregator Class Reference

Baseclass for aggregators. An aggregator is an object that aggregates multiple data in a combined datastructure or element of a small size. Examples of aggregating elements are Flows, Tree, and DNS-records.

```
#include <Aggregator.h>
```

Inheritance diagram for TAggregator:



#### Public Member Functions

- virtual uint8\_t [add](#) (void)
- virtual void [dump](#) (void)

#### Public Attributes

- int32\_t [Index](#)  
*Index of the current element.*

## Protected Attributes

- `int32_t` [Size](#)  
*Total number of used elements.*
- `int32_t` [WriteIndex](#)  
*Index to the first element to write in an array.*

### 5.1.1 Detailed Description

Baseclass for aggregators. An aggregator is an object that aggregates multiple data in a combined datastructure or element of a small size. Examples of aggregating elements are Flows, Tree, and DNS-records.

<

### 5.1.2 Member Function Documentation

5.1.2.1 `virtual uint8_t TAggregator::add ( void )` `[inline, virtual]`

Reimplemented in [TFlowAggregator](#), [TDNSHelper](#), and [THTTPHelper](#).

5.1.2.2 `virtual void TAggregator::dump ( void )` `[inline, virtual]`

adds data to the Aggregator. @ return the result of the addition:

- 1 = succesful addition in existing element(s)
- 2 = new element(s) created
- other = error in processing, data not added

### 5.1.3 Member Data Documentation

5.1.3.1 `int32_t TAggregator::Index`

Index of the current element.

5.1.3.2 `int32_t TAggregator::Size` `[protected]`

Total number of used elements.

5.1.3.3 `int32_t TAggregator::WriteIndex` `[protected]`

Index to the first element to write in an array.

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

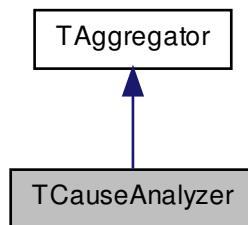
- [src/Aggregator.h](#)

## 5.2 TCauseAnalyzer Class Reference

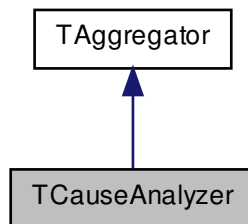
Class that manages causal trees and sorts new flows in it.

```
#include <CauseAnalyzer.h>
```

Inheritance diagram for TCauseAnalyzer:



Collaboration diagram for TCauseAnalyzer:



### Public Member Functions

- [TCauseAnalyzer](#) (void)
- [uint8\\_t add](#) (char \*name)
- [void dump](#) (char \*dm, int dest)

### Public Attributes

- char [ID](#) [256]
- char [BestID](#) [256]

### Private Attributes

- int32\_t [CauseCounter](#)
- int32\_t [DNSCauseCounter](#)
- int32\_t [DELDNSCauseCounter](#)
- int32\_t [RPTDNSCauseCounter](#)
- int32\_t [URLCauseCounter](#)
- int32\_t [DELUURLCauseCounter](#)
- int32\_t [SURLCauseCounter](#)
- int32\_t [DELSURLCauseCounter](#)
- int32\_t [HTTPCauseCounter](#)
- int32\_t [HTTPSCauseCounter](#)
- int32\_t [USERCauseCounter](#)
- int32\_t [SERVERCauseCounter](#)
- int32\_t [WhiteListCauseCounter](#)
- int32\_t [AlreadyOpenCauseCounter](#)
- int32\_t [UTreeCauseCounter](#)
- int32\_t [UnknownCauseCounter](#)
- int32\_t [DNSUnknownCauseCounter](#)
- int32\_t [ResolvedDNSUnknownCauseCounter](#)
- int32\_t [IDLOverLengthCounter](#)
- int32\_t [IDLOverTokenCounter](#)
- int64\_t [LastUserTimeStamp](#)
- int64\_t [StatDNSTimes](#) [100000]
- int32\_t [StatDNSTimesCounter](#)
- int64\_t [StatURLTimes](#) [100000]
- int32\_t [StatURLTimesCounter](#)
- int32\_t [NoNameFlowCounter](#)
- int32\_t [DNSNoNameFlowCounter](#)
- int32\_t [CauseCallCounter](#)
- int32\_t [DNSCounter](#)
- int32\_t [FoundDNSIndex](#) [10]
- int8\_t [FoundDNSCount](#)
- int64\_t [WorstDNSTime](#)
- int8\_t [WorstDNSPlace](#)
- int8\_t [BestDNSPlace](#)
- int64\_t [BestDNSTime](#)
- int64\_t [DNSDelay](#)
- int32\_t [BestURLEventIndex](#)
- int16\_t [BestURLHash](#)
- int64\_t [BestURLDelay](#)



- int32\_t [URLEventIndex](#)
- int32\_t [BestHTTPEventIndex](#)
- int32\_t [BestUserEventIndex](#)
- int64\_t [UserDelay](#)

### 5.2.1 Detailed Description

Class that manages causal trees and sorts new flows in it.

< An instance of [TCauseAnalyzer](#) creates a static array of TREE\_BUFFER\_SIZE (65535) empty trees. The FlowAggregator will call this object in case of a new flow.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 TCauseAnalyzer::TCauseAnalyzer ( void )

Constructor. During creation a fixed array of tree is created for fast storage.

### 5.2.3 Member Function Documentation

#### 5.2.3.1 uint8\_t TCauseAnalyzer::add ( char \* name )

Adds a new flow to the appropriate tree.

#### Returns

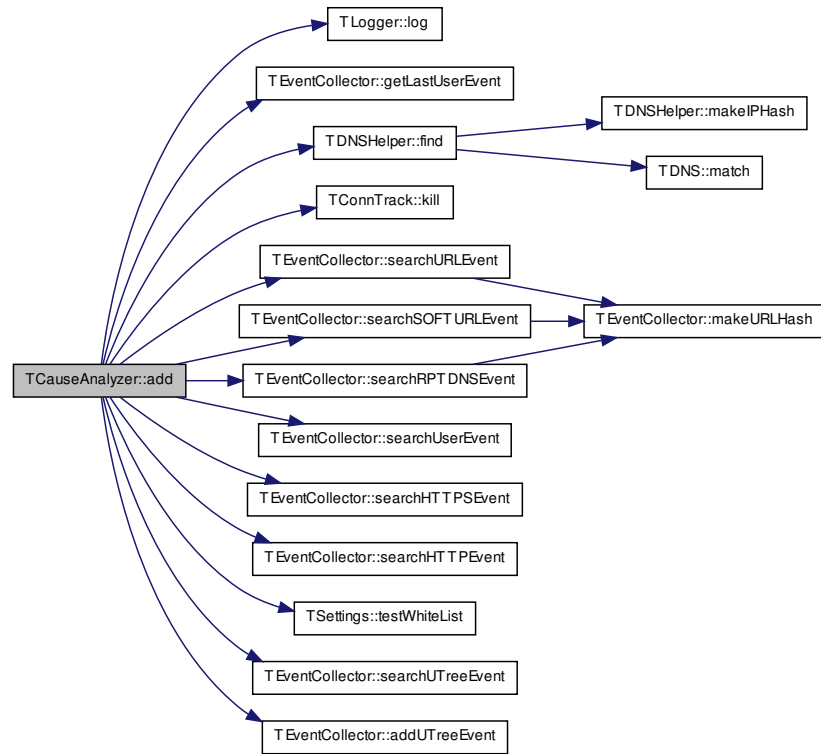
the result of the addition:

- 1 = succesful update of existing tree, Index points to this tree
- 2 = new tree created, Index points to this tree
- other = error in processing, flow not added

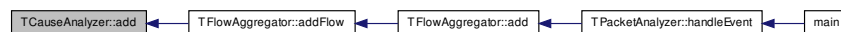
#### Parameters

<i>*name</i>	Pointer to name string in case of a DNS-reply
--------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.2.3.2 void TCauseAnalyzer::dump ( char \* dm, int dest )

Dumps the content of all non-empty trees to stdout or logfile

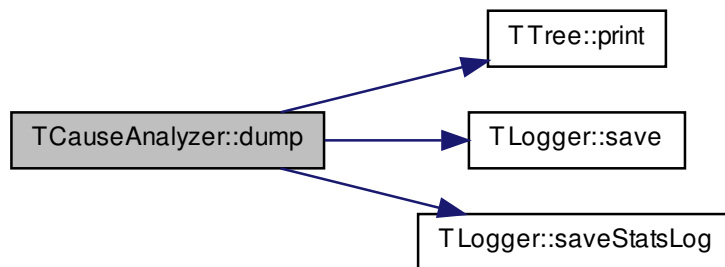
#### Returns

void

## Parameters

<i>*dm</i>	pointer to content that must be printed. NULL is print to stdout
<i>dest</i>	destination of the log: 0= to stdout, 1=to logfile

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.2.4 Member Data Documentation

5.2.4.1 `int32_t TCauseAnalyzer::AlreadyOpenCauseCounter` [private]

5.2.4.2 `int8_t TCauseAnalyzer::BestDNSPlace` [private]

5.2.4.3 `int64_t TCauseAnalyzer::BestDNSTime` [private]

5.2.4.4 `int32_t TCauseAnalyzer::BestHTTPEventIndex` [private]

5.2.4.5 `char TCauseAnalyzer::BestID[256]`

String with most recent resolved name of last analyzed flow

- 5.2.4.6 `int64_t TCauseAnalyzer::BestURLDelay` [private]
  - 5.2.4.7 `int32_t TCauseAnalyzer::BestURLEventIndex` [private]
  - 5.2.4.8 `int16_t TCauseAnalyzer::BestURLHash` [private]
  - 5.2.4.9 `int32_t TCauseAnalyzer::BestUserEventIndex` [private]
  - 5.2.4.10 `int32_t TCauseAnalyzer::CauseCallCounter` [private]
  - 5.2.4.11 `int32_t TCauseAnalyzer::CauseCounter` [private]
  - 5.2.4.12 `int32_t TCauseAnalyzer::DELDNSCauseCounter` [private]
  - 5.2.4.13 `int32_t TCauseAnalyzer::DELSURLCauseCounter` [private]
  - 5.2.4.14 `int32_t TCauseAnalyzer::DELURLCauseCounter` [private]
  - 5.2.4.15 `int32_t TCauseAnalyzer::DNSCauseCounter` [private]
  - 5.2.4.16 `int32_t TCauseAnalyzer::DNSCounter` [private]
  - 5.2.4.17 `int64_t TCauseAnalyzer::DNSDelay` [private]
  - 5.2.4.18 `int32_t TCauseAnalyzer::DNSNoNameFlowCounter` [private]
  - 5.2.4.19 `int32_t TCauseAnalyzer::DNSUnknownCauseCounter` [private]
  - 5.2.4.20 `int8_t TCauseAnalyzer::FoundDNSCount` [private]
- Number of valid DNS records
- 5.2.4.21 `int32_t TCauseAnalyzer::FoundDNSIndex[10]` [private]
- DNS-indices that could match
- 5.2.4.22 `int32_t TCauseAnalyzer::HTTPCauseCounter` [private]
  - 5.2.4.23 `int32_t TCauseAnalyzer::HTTPSCauseCounter` [private]
  - 5.2.4.24 `char TCauseAnalyzer::ID[256]`
- String with IP or name of last analyzed flow
- 5.2.4.25 `int32_t TCauseAnalyzer::IDLOverLengthCounter` [private]

5.2.4.26 `int32_t TCauseAnalyzer::IDLOverTokenCounter` [private]

5.2.4.27 `int64_t TCauseAnalyzer::LastUserTimeStamp` [private]

5.2.4.28 `int32_t TCauseAnalyzer::NoNameFlowCounter` [private]

For registration of response times

5.2.4.29 `int32_t TCauseAnalyzer::ResolvedDNSUnknownCauseCounter`  
[private]

5.2.4.30 `int32_t TCauseAnalyzer::RPTDNSCauseCounter` [private]

5.2.4.31 `int32_t TCauseAnalyzer::SERVERCauseCounter` [private]

5.2.4.32 `int64_t TCauseAnalyzer::StatDNSTimes[100000]` [private]

For registration of response times

5.2.4.33 `int32_t TCauseAnalyzer::StatDNSTimesCounter` [private]

For registration of response times

5.2.4.34 `int64_t TCauseAnalyzer::StatURLTimes[100000]` [private]

For registration of response times

5.2.4.35 `int32_t TCauseAnalyzer::StatURLTimesCounter` [private]

For registration of response times

5.2.4.36 `int32_t TCauseAnalyzer::SURLCauseCounter` [private]

5.2.4.37 `int32_t TCauseAnalyzer::UnknownCauseCounter` [private]

5.2.4.38 `int32_t TCauseAnalyzer::URLCauseCounter` [private]

5.2.4.39 `int32_t TCauseAnalyzer::URLEventIndex` [private]

5.2.4.40 `int32_t TCauseAnalyzer::USERCauseCounter` [private]

5.2.4.41 `int64_t TCauseAnalyzer::UserDelay` [private]

5.2.4.42 `int32_t TCauseAnalyzer::UTreeCauseCounter` [private]

5.2.4.43 `int32_t TCauseAnalyzer::WhiteListCauseCounter` [private]

5.2.4.44 `int8_t TCauseAnalyzer::WorstDNSPlace` [private]

5.2.4.45 `int64_t TCauseAnalyzer::WorstDNSTime` [private]

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

- [src/CauseAnalyzer.h](#)
- [src/CauseAnalyzer.cpp](#)

## 5.3 TConnTrack Class Reference

Class that can kill an established connection.

```
#include <ConnTrack.h>
```

### Public Member Functions

- [TConnTrack](#) (char \*ip)
- int [kill](#) (uint32\_t src, uint32\_t dst, uint8\_t prot, uint16\_t psrc, uint16\_t pdst)
- int [kill](#) (uint32\_t flowindex)

#### 5.3.1 Detailed Description

Class that can kill an established connection.

<

#### 5.3.2 Constructor & Destructor Documentation

##### 5.3.2.1 TConnTrack::TConnTrack ( char \* ip )

Constructor. Initialises Firewall

##### Parameters

<i>ip</i>	local IP-address
-----------	------------------

#### 5.3.3 Member Function Documentation

##### 5.3.3.1 int TConnTrack::kill ( uint32\_t src, uint32\_t dst, uint8\_t prot, uint16\_t psrc, uint16\_t pdst )

Kills a flow by specified 5-tuple

Returns

0=success, -1=error

Parameters

<i>src</i>	source IP-address (=local IP-address)
<i>dst</i>	dest IP-address (=remote IP-address)
<i>prot</i>	protocol (only 6=TCP or 17=UDP)
<i>psrc</i>	source port (=local port)
<i>pdst</i>	dest port (=remote port)

Here is the caller graph for this function:



5.3.3.2 int TConnTrack::kill ( uint32\_t flowindex )

Kills a flow by specified flowindex

Returns

0=success, -1=error

Parameters

<i>flowindex</i>	
------------------	--

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

- [src/ConnTrack.h](#)
- [src/ConnTrack.cpp](#)

5.4 TDNS Class Reference

Storage class for DNS-data. Objects of this class are managed by DNSHelper.

```
#include <DNS.h>
```

Public Member Functions

- [TDNS](#) (void)

- void [clear](#) (void)
- void [set](#) (int32\_t f, int64\_t ftime, uint32\_t ipaddr, uint32\_t t, char \*nm, char \*cm)
- int [match](#) (uint32\_t ip, char \*name)
- void [print](#) (char \*content)

## Public Attributes

- int64\_t [TimeStamp](#)  
*Timestamp of the recieved data.*
- uint32\_t [IP](#)  
*IP-address.*
- char [NAME](#) [MAXDOMAINNAMELENGTH]  
*Domain name of the original Query.*
- char [CNAME](#) [MAXDOMAINNAMELENGTH]  
*Domain name in the RR (possibly a CNAME)*
- uint32\_t [TTL](#)  
*Time To Live in seconds.*
- uint8\_t [Resolved](#)  
*Boolean 0= not used, 1=at least 1 received IP is used.*
- int32\_t [FlowIndex](#)  
*Index to the most recent DNS Flow that delivered this information.*
- int32\_t [NextDNSIndex](#)  
*Links to a next DNS-record by an index (unidirectional linked list).*

### 5.4.1 Detailed Description

Storage class for DNS-data. Objects of this class are managed by DNSHelper.

<

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 TDNS::TDNS ( void )

The constructor. Resets all the data-members



Here is the call graph for this function:



### 5.4.3 Member Function Documentation

#### 5.4.3.1 void TDNS::clear ( void )

Clears all the data-members, except NextDNSIndex

##### Returns

void

Here is the caller graph for this function:



#### 5.4.3.2 int TDNS::match ( uint32\_t ip, char \* name )

Matches IP and Name of DNS-record. If \*name is NULL only the IP-address is used in the search

##### Returns

0=no match, 1=IP match, 2=NAME match, 3=CNAME match

##### Parameters

<i>ip</i>	IP-address
<i>*name</i>	Pointer to Name

Here is the caller graph for this function:



5.4.3.3 void TDNS::print ( char \* content )

Prints a flow in readable format to stdout

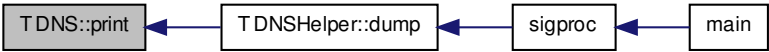
Returns

void

Parameters

<i>*content</i>	pointer to content that must be printed. NULL is print to stdout
-----------------	--

Here is the caller graph for this function:



5.4.3.4 void TDNS::set ( int32\_t f, int64\_t ftime, uint32\_t ipaddr, uint32\_t t, char \* nm, char \* cm )

Sets a DNS-object except for the linked list pointer

Returns

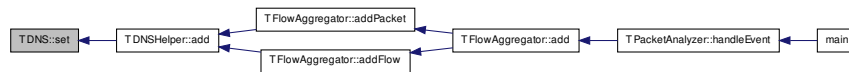
void

Parameters

<i>f</i>	Index to the DNS Flow that delivered this information
<i>ftime</i>	First time the IP-address was referenced in a DNS A record
<i>ipaddr</i>	IP-address of the RR
<i>t</i>	TTL in the RR
<i>nm</i>	Name in the Query

<i>cm</i>	Name in the Answer RR (probably a CNAME or the same as the Query Name)
-----------	--

Here is the caller graph for this function:



#### 5.4.4 Member Data Documentation

##### 5.4.4.1 char TDNS::CNAME[MAXDOMAINNAMELENGTH]

Domain name in the RR (possibly a CNAME)

##### 5.4.4.2 int32\_t TDNS::FlowIndex

Index to the most recent DNS Flow that delivered this information.

##### 5.4.4.3 uint32\_t TDNS::IP

IP-address.

##### 5.4.4.4 char TDNS::NAME[MAXDOMAINNAMELENGTH]

Domain name of the original Query.

##### 5.4.4.5 int32\_t TDNS::NextDNSIndex

Links to a next DNS-record by an index (unidirectional linked list).

##### 5.4.4.6 uint8\_t TDNS::Resolved

Boolean 0= not used, 1=at least 1 received IP is used.

##### 5.4.4.7 int64\_t TDNS::TimeStamp

Timestamp of the recieved data.

#### 5.4.4.8 uint32\_t TDNS::TTL

Time To Live in seconds.

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

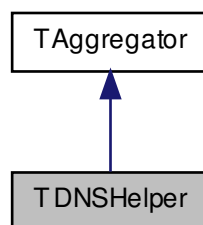
- src/[DNS.h](#)
- src/[DNS.cpp](#)

## 5.5 TDNSHelper Class Reference

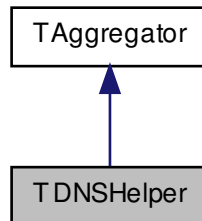
Class specialized in aggregating DNS events. It only processes replies from egress UDP DNS-flows. DNS objects are stored as a chained hashtable of unidirectional linked lists. The hashtable is based on a modulo 256 sum of the IP-address. This results in a fast lookup by IP-address. Newer records with the same name and IP-address are replaced. TODO: cleanup-routine that checks one record at a time on expiry.

```
#include <DNSHelper.h>
```

Inheritance diagram for TDNSHelper:



Collaboration diagram for TDNSHelper:



### Public Member Functions

- [TDNSHelper](#) (void)
- uint8\_t [add](#) (void)
- char \* [getQueryName](#) (void)
- uint8\_t [deleteRecord](#) (uint32\_t i)
- int [find](#) (uint32\_t ip, char \*name)
- uint16\_t [makeIPHash](#) (uint32\_t ip)
- void [dump](#) (char \*content)

### Private Member Functions

- uint16\_t [parseName](#) (uint16\_t offset)
- void [clearQueryName](#) (void)
- uint8\_t [addToQueryName](#) (char s)
- void [clearAnswerName](#) (void)
- uint8\_t [addToAnswerName](#) (char s)

### Private Attributes

- const uint8\_t \* [Packet](#)  
*packet that starts from DNS start*
- char [QueryName](#) [256]  
*string for parsed query name*
- char [AnswerName](#) [256]  
*string for parsed answer name*
- uint16\_t [QueryNameIndex](#)  
*write-pointer in QueryString*

- uint16\_t [AnswerNameIndex](#)

*write-pointer in AnswerString*

- uint16\_t [Length](#)

*Length of DNS-header and payload.*

- int32\_t [DNSTable](#) [[DNSHASHSIZE](#)]

*Hashtable: Array of indexes to the buckets that belong to the hashtable.*

### 5.5.1 Detailed Description

Class specialized in aggregating DNS events. It only processes replies from egress UDP DNS-flows. DNS objects are stored as a chained hashtable of unidirectional linked lists. The hashtable is based on a modulo 256 sum of the IP-address. This results in a fast lookup by IP-address. Newer records with the same name and IP-address are replaced. TODO: cleanup-routine that checks one record at a time on expiry.

<

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 TDNSHelper::TDNSHelper ( void )

The constructor. Resets all the data-members, including NextFlow

### 5.5.3 Member Function Documentation

#### 5.5.3.1 uint8\_t TDNSHelper::add ( void ) [virtual]

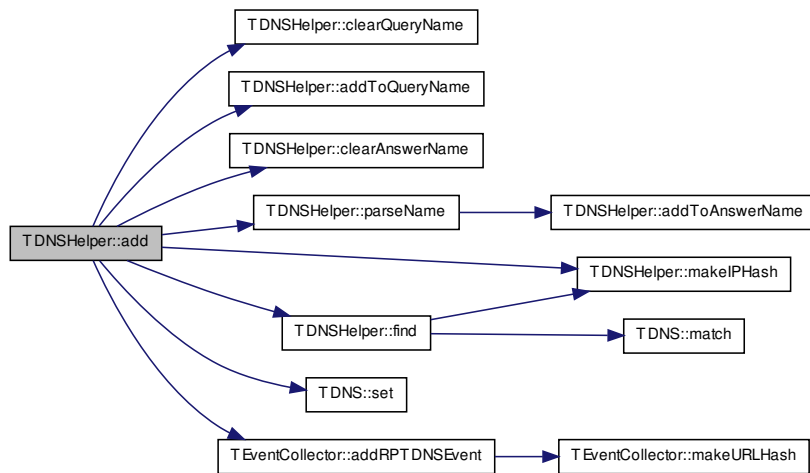
Processes a DNS-response.

**Returns**

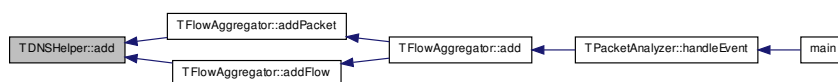
1=success: data updated in existing records, 2=success: new DNS-record(s) added, 3+ = error, 0=ingress

Reimplemented from [TAggregator](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.5.3.2 uint8\_t TDNSHelper::addToAnswerName ( char s ) [private]

Copies a character to the AnswerName array.

**Returns**

1=success, 0=array full

## Parameters

s	character
---	-----------

Here is the caller graph for this function:



### 5.5.3.3 uint8\_t TDNSHelper::addToQueryName ( char s ) [private]

Copies a character to the QueryName array.

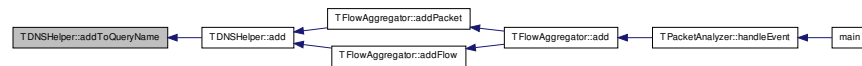
## Returns

1=success, 0=array full

## Parameters

s	character
---	-----------

Here is the caller graph for this function:



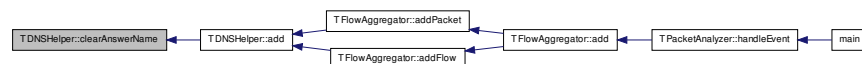
### 5.5.3.4 void TDNSHelper::clearAnswerName ( void ) [private]

Clears AnswerName array.

## Returns

void

Here is the caller graph for this function:





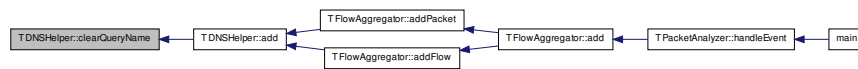
### 5.5.3.5 void TDNSHelper::clearQueryName ( void ) [private]

Clears QueryName array.

#### Returns

void

Here is the caller graph for this function:



### 5.5.3.6 uint8\_t TDNSHelper::deleteRecord ( int32\_t i )

Frees a DNS-record.

#### Returns

1=ok

#### Parameters

<i>i</i>	Index of the DNS-record in the static array of DNS-records.
----------	---

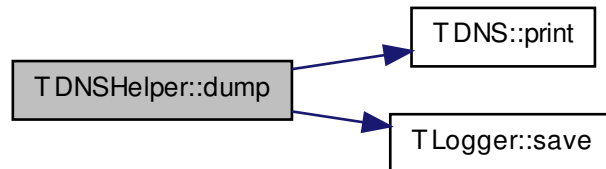
### 5.5.3.7 void TDNSHelper::dump ( char \* content )

Dumps the content of all DNS-records to stdout return void

#### Parameters

<i>*content</i>	pointer to content that must be printed. NULL is print to stdout
-----------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.5.3.8 int TDNSHelper::find ( uint32\_t ip, char \* name )

Finds a DNS-record by specified IP-address and name. If name=NULL only IP-match.

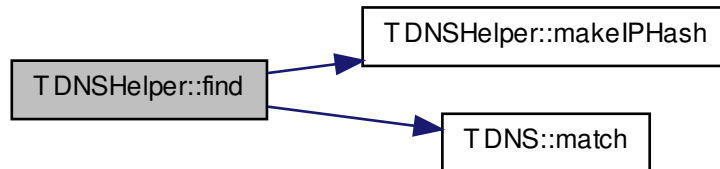
##### Returns

status, 0=no match-no items left, 1=IP-match, 2=name match, 3=cname match,  
Index points to the last queried item

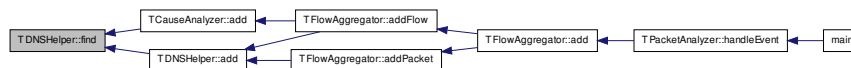
##### Parameters

<i>ip</i>	IP-address
<i>name</i>	Name of the Query, if NULL only the ip-adress is used in the search

Here is the call graph for this function:



Here is the caller graph for this function:



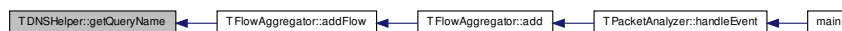
#### 5.5.3.9 `char * TDNSHelper::getQueryName ( void )`

Gets the Queryname. Only to be called immediatly after add.

##### Returns

Pointer to string with most recent queryname

Here is the caller graph for this function:



#### 5.5.3.10 `uint16_t TDNSHelper::makeIPHash ( uint32_t ip )`

Calculates a modulo HASHSIZE of the IP-address

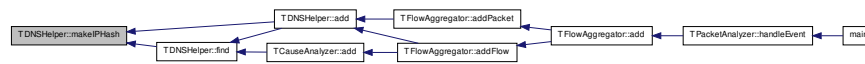
This can be used for organising a fast lookup by a hash-table.

The hashspace is set by `DNSHASHSIZE` (default=1024).

**Returns**

HashValue

Here is the caller graph for this function:



### 5.5.3.11 uint16\_t TDNSHelper::parseName ( uint16\_t offset ) [private]

Parses a name in an answer from the DNS payload. In case of a pointer it is recursively called.

**Returns**

0=overflow, in all other cases it points to the next field in the packet after the name.

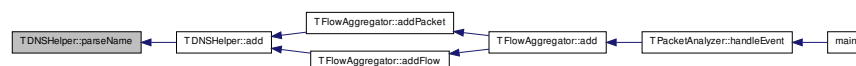
**Parameters**

<i>offset</i>	present index in the packet
---------------	-----------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.5.4 Member Data Documentation

**5.5.4.1** `char TDNSHelper::AnswerName[256]` `[private]`

string for parsed answer name

**5.5.4.2** `uint16_t TDNSHelper::AnswerNameIndex` `[private]`

write-pointer in AnswerString

**5.5.4.3** `int32_t TDNSHelper::DNSTable[DNSHASHSIZE]` `[private]`

Hashtable: Array of indexes to the buckets that belong to the hashtable.

**5.5.4.4** `uint16_t TDNSHelper::Length` `[private]`

Length of DNS-header and payload.

**5.5.4.5** `const uint8_t* TDNSHelper::Packet` `[private]`

packet that starts from DNS start

**5.5.4.6** `char TDNSHelper::QueryName[256]` `[private]`

string for parsed query name

**5.5.4.7** `uint16_t TDNSHelper::QueryNameIndex` `[private]`

write-pointer in QueryString

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

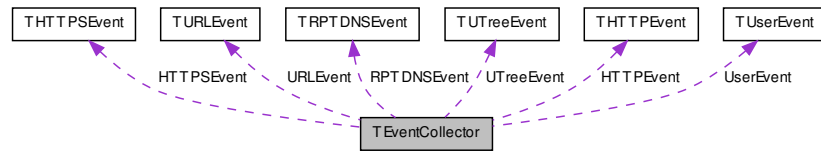
- [src/DNSHelper.h](#)
- [src/DNSHelper.cpp](#)

## 5.6 TEventCollector Class Reference

Class specialized in aggregating events that can cause new traffic. Every potential event is buffered here. Hashing is used for fast lookup. FIFO's are used to prevent overflow.

```
#include <EventCollector.h>
```

Collaboration diagram for TEventCollector:



## Public Member Functions

- [TEventCollector](#) (void)
- void [addUserEvent](#) (int64\_t timestamp, uint8\_t eventcode, char \*process)
- void [addHTTPEvent](#) (int64\_t timestamp, uint8\_t eventcode, uint32\_t flowindex)
- void [removeHTTPEvent](#) (int64\_t timestamp, int32\_t flowindex)
- void [addHTTPSEvent](#) (int64\_t timestamp, uint8\_t eventcode, uint32\_t flowindex)
- void [addUTreeEvent](#) (int64\_t timestamp, uint32\_t treeindex)
- void [removeHTTPSEvent](#) (int64\_t timestamp, int32\_t flowindex)
- void [addURLEvent](#) (int64\_t timestamp, char \*url, int32\_t flowindex)
- void [addRPTDNSEvent](#) (int64\_t timestamp, char \*name, int32\_t flowindex)
- int32\_t [searchUserEvent](#) (int64\_t \*delay)
- int32\_t [searchHTTPEvent](#) (int64\_t \*delay, int prot)
- int32\_t [searchHTTPSEvent](#) (int64\_t \*delay, int prot)
- int32\_t [searchURLEvent](#) (char \*url, int64\_t \*delay, int32\_t index)
- int32\_t [searchUTreeEvent](#) (int64\_t \*delay)
- int32\_t [searchSOFTURLEvent](#) (char \*url, int64\_t \*delay, int32\_t index)
- int32\_t [searchRPTDNSEvent](#) (char \*name, int64\_t \*delay)
- int64\_t [getLastUserEvent](#) (void)
- void [makeIPHash](#) (uint32\_t ip)
- void [makeURLHash](#) (char \*url)
- void [dump](#) (char \*content, int dest)

## Public Attributes

- [TUserEvent](#) `UserEvent` [`EVENTBUFFERSIZE`]  
*FIFO of user events.*
- [THTTPEvent](#) `HTTPEvent` [`EVENTBUFFERSIZE`]  
*FIFO of generic HTTP events.*
- [THTTPSEvent](#) `HTTPSEvent` [`EVENTBUFFERSIZE`]  
*FIFO of generic HTTPS events.*
- [TUTreeEvent](#) `UTreeEvent` [`EVENTBUFFERSIZE`]  
*FIFO of Trees with Unknown cause.*

- [TURLEvent URLEvent \[URLEVENTHASHSIZE\]\[URLEVENTBUFFERSIZE\]](#)  
*Hashtable with URLEVENTHASHSIZE buckets. Each bucket is a FIFO.*
- [TRPTDNSEvent RPTDNSEvent \[URLEVENTHASHSIZE\]\[RPTDNSEVENTBUFFERSIZE\]](#)  
*Hashtable with URLEVENTHASHSIZE buckets. Each bucket is a FIFO.*
- `int16_t` [Hash](#)
- `int64_t` [AggregatedWindowTime](#)
- `int64_t` [WindowEndTime](#)
- `int32_t` [UserEventCount](#)
- `int32_t` [HTTPEventCount](#)
- `int32_t` [HTTPSEventCount](#)
- `int32_t` [UTreeEventCount](#)

### Private Attributes

- `int32_t` [UserEventIndex](#)  
*index to most recent event or -1 if empty*
- `int32_t` [HTTPEventIndex](#)  
*index to most recent event or -1 if empty*
- `int32_t` [HTTPSEventIndex](#)  
*index to most recent event or -1 if empty*
- `int32_t` [UTreeEventIndex](#)  
*index to most recent event or -1 if empty*
- `int32_t` [URLEventIndex \[URLEVENTHASHSIZE\]](#)  
*array of indices to most recent URL events or -1 if empty*
- `int32_t` [RPTDNSEventIndex \[URLEVENTHASHSIZE\]](#)  
*array of indices to most recent REPEAT DNS events or -1 if empty*

#### 5.6.1 Detailed Description

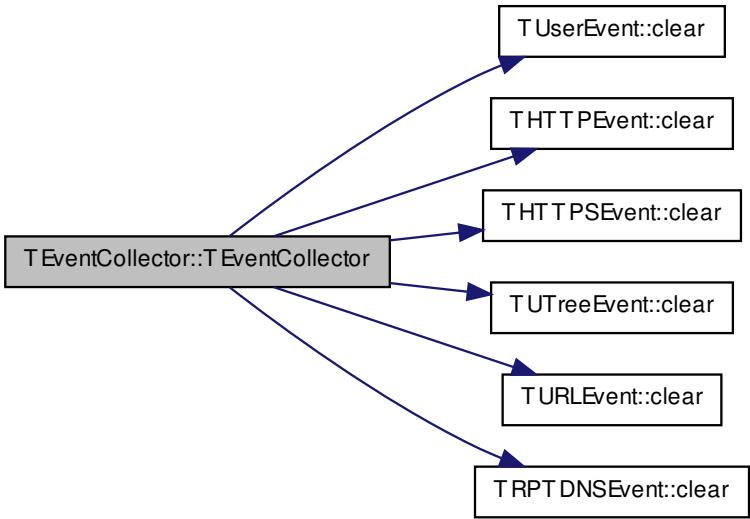
Class specialized in aggregating events that can cause new traffic. Every potential event is buffered here. Hashing is used for fast lookup. FIFO's are used to prevent overflow.

<

#### 5.6.2 Constructor & Destructor Documentation

5.6.2.1 TEventCollector::TEventCollector ( void )

Here is the call graph for this function:



5.6.3 Member Function Documentation

5.6.3.1 void TEventCollector::addHTTPEvent ( int64\_t timestamp, uint8\_t eventcode, uint32\_t flowindex )

Adds a HTTP event

Returns

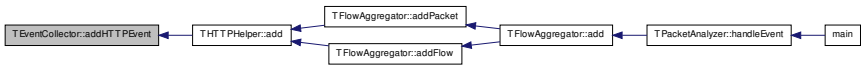
void

Parameters

timestamp	
eventcode	Describes the type of event: 0=non trivial data received, 1=push flag received
flowindex	Index to the HTTP flow



Here is the caller graph for this function:



5.6.3.2 void TEventCollector::addHTTPSEvent ( int64\_t timestamp, uint8\_t eventcode, uint32\_t flowindex )

Adds a HTTP event

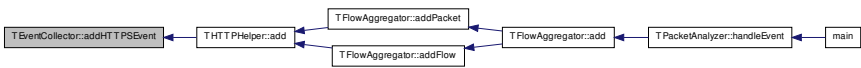
Returns

void

Parameters

timestamp	
eventcode	Describes the type of event: 0=non trivial data received, 1=push flag received
flowindex	Index to the HTTP flow

Here is the caller graph for this function:



5.6.3.3 void TEventCollector::addRPTDNSEvent ( int64\_t timestamp, char \* name, int32\_t flowindex )

Adds a REPEAT DNS event

Returns

void

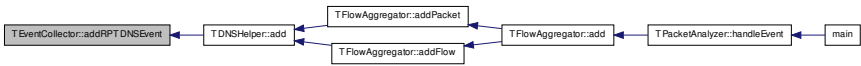
Parameters

timestamp	
*name	Pointer to the string that contains the query name
flowindex	Index to the DNS flow

Here is the call graph for this function:



Here is the caller graph for this function:



5.6.3.4 void TEventCollector::addURLEvent ( int64\_t timestamp, char \* url, int32\_t flowindex )

Adds a HTTP URL event

Returns

void

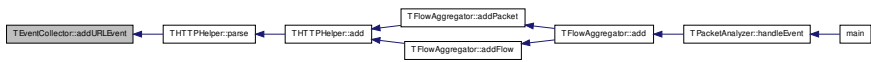
Parameters

timestamp	
*url	Pointer to the string that contains the received URL
flowindex	Index to the HTTP flow

Here is the call graph for this function:



Here is the caller graph for this function:



5.6.3.5 void TEventCollector::addUserEvent ( int64\_t timestamp, uint8\_t eventcode, char \* process )

Adds a user event

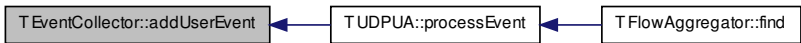
Returns

void

Parameters

timestamp	
eventcode	Describes the type of event: 0=other, 1=F5, 2=LMB, 3=Enter
*process	Pointer to a string that describes the geerating process (if supported by the agent)

Here is the caller graph for this function:



5.6.3.6 void TEventCollector::addUTreeEvent ( int64\_t timestamp, uint32\_t treeindex )

Adds an Unknown cause Tree event

Returns

void

Parameters

timestamp	
treeindex	Index to the Tree

Here is the caller graph for this function:



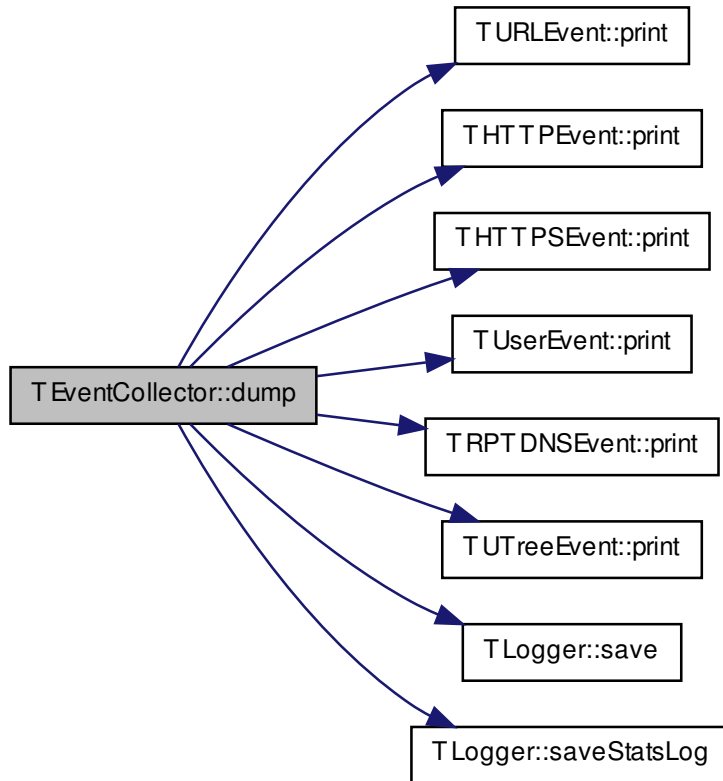
#### 5.6.3.7 void TEventCollector::dump ( char \* *content*, int *dest* )

Dumps the content of all Events to stdout return void

##### Parameters

* <i>content</i>	pointer to content that must be printed. NULL is print to stdout
<i>dest</i>	destination of the log: 0= to stdout, 1=to logfile

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.6.3.8 int64\_t TEventCollector::getLastUserEvent ( void )

Gives timestamp of last enter/mousclick/F5 event or zero if none.

#### Returns

timestamp of last enter/mousclick/F5 or 0 if none

Here is the caller graph for this function:



### 5.6.3.9 void TEventCollector::makeIPHash ( uint32\_t ip )

Calculates the hash of an IP-address

#### Returns

hash

#### Parameters

<i>ip</i>	IP-adress
-----------	-----------

### 5.6.3.10 void TEventCollector::makeURLHash ( char \* url )

Calculates the hash of an URL or name

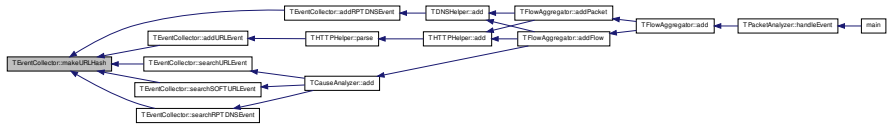
#### Returns

hash

#### Parameters

<i>*url</i>	Pointer to name string
-------------	------------------------

Here is the caller graph for this function:



5.6.3.11 void TEventCollector::removeHTTPEvent ( int64\_t timestamp, int32\_t flowindex )

Removes a HTTP event, by setting the Eventcode to CAUSE\_WITHDRAWN

Returns

void

Parameters

timestamp	
flowindex	Index to the HTTP flow

5.6.3.12 void TEventCollector::removeHTTPSEvent ( int64\_t timestamp, int32\_t flowindex )

Removes a HTTP event, by setting the Eventcode to CAUSE\_WITHDRAWN

Returns

void

Parameters

timestamp	
flowindex	Index to the HTTP flow

5.6.3.13 int32\_t TEventCollector::searchHTTPEvent ( int64\_t \* delay, int prot )

Delivers a HTTP event, starting with the most recent one and counting back on every call.

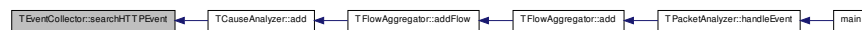
Returns

index to most Event or -1 if empty

## Parameters

<i>*delay</i>	max search time to lookup back in history, returns the time between found event and present time
<i>prot</i>	80=http, 443=https

Here is the caller graph for this function:



#### 5.6.3.14 int32\_t TEventCollector::searchHTTPSEvent ( int64\_t \* *delay*, int *prot* )

Delivers a HTTP event, starting with the most recent one and counting back on every call.

## Returns

index to most Event or -1 if empty

## Parameters

<i>*delay</i>	max search time to lookup back in history, returns the time between found event and present time
<i>prot</i>	80=http, 443=https

Here is the caller graph for this function:



#### 5.6.3.15 int32\_t TEventCollector::searchRPTDNSEvent ( char \* *name*, int64\_t \* *delay* )

Delivers the flowindex of a REPEAT DNS event, including the name and delay, by walking back in history.

## Returns

index to FlowIndex that created the event



## Parameters

<i>*name</i>	Pointer to the string that must match the query name
<i>*delay</i>	max search time to lookup back in history, returns the time between found event and present time

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.6.3.16 int32\_t TEventCollector::searchSOFTURLEvent ( char \* url, int64\_t \* delay, int32\_t index )

Delivers a SOFTHTTP event by URL substring, by walking back in history.

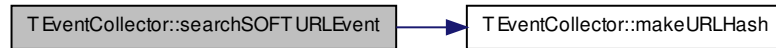
## Returns

index to most Event or -1 if empty

## Parameters

<i>*url</i>	Pointer to the string that must match the URL
<i>*delay</i>	max search time to lookup back in history, returns the time between found event and present time
<i>index</i>	Last found event, used for search continuation, -1 is start from the most recent entry

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.6.3.17 int32\_t TEventCollector::searchURLEvent ( char \* url, int64\_t \* delay, int32\_t index )

Delivers a HTTP event by URL, by walking back in history.

##### Returns

index to most Event or -1 if empty

##### Parameters

<i>*url</i>	Pointer to the string that must match the URL
<i>*delay</i>	max search time to lookup back in history, returns the time between found event and present time
<i>index</i>	Pointer to Index of Last found event, -1=start new search or as return: nothing found

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.6.3.18 int32\_t TEventCollector::searchUserEvent ( int64\_t \* delay )

Delivers a DNS event, starting with the most recent one and counting back on every call.

##### Returns

index to most Event or -1 if empty

##### Parameters

<i>*delay</i>	max search time to lookup back in history, returns the time between found event and present time
---------------	--

Here is the caller graph for this function:



#### 5.6.3.19 int32\_t TEventCollector::searchUTreeEvent ( int64\_t \* delay )

Delivers the most recent unknown cause tree event, by walking back in history.

##### Returns

index to most Event or -1 if empty

##### Parameters

<i>*delay</i>	pointer to max search time to lookup back in history, returns the time between found event and present time
---------------	---

Here is the caller graph for this function:



## 5.6.4 Member Data Documentation

5.6.4.1 `int64_t TEventCollector::AggregatedWindowTime`

5.6.4.2 `int16_t TEventCollector::Hash`

5.6.4.3 `THTTPEvent TEventCollector::HTTPEvent[EVENTBUFFERSIZE]`

FIFO of generic HTTP events.

5.6.4.4 `int32_t TEventCollector::HTTPEventCount`

5.6.4.5 `int32_t TEventCollector::HTTPEventIndex` `[private]`

index to most recent event or -1 if empty

5.6.4.6 `THTTPSEvent TEventCollector::HTTPSEvent[EVENTBUFFERSIZE]`

FIFO of generic HTTPS events.

5.6.4.7 `int32_t TEventCollector::HTTPSEventCount`

5.6.4.8 `int32_t TEventCollector::HTTPSEventIndex` `[private]`

index to most recent event or -1 if empty

5.6.4.9 `TRPTDNSEvent TEventCollector::RPTDNSEvent[URLEVENTHASHSIZE][RPTDNSEVENTBUFFERSIZE]`

Hastable with URLEVENTHASHSIZE buckets. Each bucket is a FIFO.

5.6.4.10 `int32_t TEventCollector::RPTDNSEventIndex[URLEVENTHASHSIZE]`  
`[private]`

array of indices to most recent REPEAT DNS events or -1 if empty

#### 5.6.4.11 `TURLEvent TEventCollector::URLEvent[URLEVENTHASHSIZE][URLEVENTBUFFERSIZE]`

Hastable with URLEVENTHASHSIZE buckets. Each bucket is a FIFO.

#### 5.6.4.12 `int32_t TEventCollector::URLEventIndex[URLEVENTHASHSIZE]` `[private]`

array of indices to most recent URL events or -1 if empty

#### 5.6.4.13 `TUserEvent TEventCollector::UserEvent[EVENTBUFFERSIZE]`

FIFO of user events.

#### 5.6.4.14 `int32_t TEventCollector::UserEventCount`

#### 5.6.4.15 `int32_t TEventCollector::UserEventIndex` `[private]`

index to most recent event or -1 if empty

#### 5.6.4.16 `TUTreeEvent TEventCollector::UTreeEvent[EVENTBUFFERSIZE]`

FIFO of Trees with Unknown cause.

#### 5.6.4.17 `int32_t TEventCollector::UTreeEventCount`

#### 5.6.4.18 `int32_t TEventCollector::UTreeEventIndex` `[private]`

index to most recent event or -1 if empty

#### 5.6.4.19 `int64_t TEventCollector::WindowEndTime`

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

- [src/EventCollector.h](#)
- [src/EventCollector.cpp](#)

## 5.7 TFlow Class Reference

Storage class for a bidirectional flow.

```
#include <Flow.h>
```

## Public Member Functions

- [TFlow](#) (void)
- void [clear](#) (void)
- uint16\_t [getHash](#) (void)
- uint8\_t [match](#) (uint8\_t p, uint32\_t lIP, uint32\_t rIP, uint16\_t lP, uint16\_t rP, uint16\_t id)
- void [print](#) (char \*content)

## Public Attributes

- int32\_t [NextFlowIndex](#)  
*Links to a next flow by an index (a kind of linked list) -1 = no next flow.*
- uint8\_t [Status](#)  
*Status is the current status of the flow: 0=empty, 1=aggregated from 1 packet, 2=aggregated from more packets.*
- int64\_t [StartTime](#)  
*Time of first packet in Flow in useconds since 1970.*
- int64\_t [StopTime](#)  
*Time of last packet in Flow in useconds since 1970.*
- int32\_t [NumberOfTransmittedBytes](#)  
*Aggregated number of transmitted bytes.*
- int32\_t [NumberOfReceivedBytes](#)  
*Aggregated number of transmitted bytes.*
- int32\_t [NumberOfTransmittedPackets](#)  
*Aggregated number of transmitted bytes.*
- int32\_t [NumberOfReceivedPackets](#)  
*Aggregated number of transmitted bytes.*
- uint8\_t [Direction](#)  
*Flow direction: UNDEFINED(0), INGRESS(1), EGRESS(2)*
- uint8\_t [Protocol](#)  
*1=ICMP, 6=TCP, 17=UDP, 255=IP-other*
- uint32\_t [LocalIP](#)  
*IP-address of the observed local computer.*
- uint32\_t [RemoteIP](#)  
*IP-address of the remote computer.*
- uint16\_t [LocalPort](#)  
*UDP/TCP=Port, ICMP=identifier, Other=0.*
- uint16\_t [RemotePort](#)  
*UDP/TCP=Port ICMP=sequence number, Other=0.*
- uint16\_t [Identification](#)  
*Identification field (only defined for DNS, 0 for other protocols)*
- uint8\_t [TCPFlag](#)  
*Aggregated TCP-Flags if TCP-Flow.*

- uint32\_t [LocalSEQ](#)  
*Local SEQ (0 if undefined)*
- uint32\_t [RemoteSEQ](#)  
*Remote expected SEQ (0 if undefined)*
- int32\_t [DNSIndex](#)  
*points to DNS-name in a STRING-table, -1 = no DNS*
- int32\_t [HTTPIndex](#)  
*if HTTP-flow, this index points to a special object that identifies triggers of new flows,  
0 = no HTTP-traffic*
- int32\_t [TreeIndex](#)  
*id of the flow tree (tree id)*
- int32\_t [ParentFlow](#)  
*id of parent flow in the tree, 0=no parents (root flow)*
- uint8\_t [Resolver](#)  
*bit 0..3=resolver protocol (0=nothing, 1=DNS, 2..15=spare, bit 7=1=resolved)*
- uint8\_t [Cause](#)  
*Cause of the Flow.*
- uint8\_t [CauseReliability](#)  
*Reliability of the cause.*
- uint64\_t [CausalTime](#)  
*Timestamp of the cause that created this flow.*

### 5.7.1 Detailed Description

Storage class for a bidirectional flow.

<

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 TFlow::TFlow ( void )

The constructor. Resets all the data-members, including NextFlow

Here is the call graph for this function:



### 5.7.3 Member Function Documentation

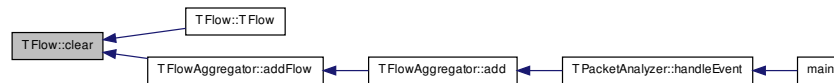
#### 5.7.3.1 void TFlow::clear ( void )

Clears all the data-members, except NextFlow

##### Returns

void

Here is the caller graph for this function:



#### 5.7.3.2 uint16\_t TFlow::getHash ( void )

Calculates a 10-bit hash by the XOR-sum of SourceIP, DestIP, Sourceport, DestPort and Identification.

This can be used for organising a fast lookup by a hash-table.

The value of hashes is set by FLOWHASHSIZE (default=1024).

##### Returns

HashValue

Here is the caller graph for this function:



#### 5.7.3.3 uint8\_t TFlow::match ( uint8\_t p, uint32\_t lIP, uint32\_t rIP, uint16\_t lP, uint16\_t rP, uint16\_t id )

Matches a flow by its addresses and port-numbers and protocol.



**Returns**

1 if matches or 251-255 if it does not match (completely).

**Parameters**

<i>p</i>	Protocol number: 1=ICMP, 6=TCP, 17=UDP, 255=IP-other
<i>lIP</i>	Local Ip-address
<i>rIP</i>	Remote Ip-address
<i>lP</i>	Local Port
<i>rP</i>	Remote Port
<i>id</i>	Identification (DNS defined, 0 for other protocols)

**5.7.3.4 void TFlow::print ( char \* content )**

Prints a flow in readable format to a string

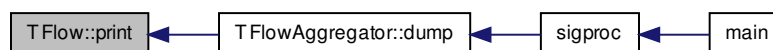
**Returns**

void

**Parameters**

<i>*content</i>	pointer to content that must be printed.
-----------------	--

Here is the caller graph for this function:

**5.7.4 Member Data Documentation****5.7.4.1 uint64\_t TFlow::CausalTime**

Timestamp of the cause that created this flow.

**5.7.4.2 uint8\_t TFlow::Cause**

Cause of the Flow.

#### 5.7.4.3 uint8\_t TFlow::CauseReliability

Reliability of the cause.

#### 5.7.4.4 uint8\_t TFlow::Direction

Flow direction: UNDEFINED(0), [INGRESS\(1\)](#), [EGRESS\(2\)](#)

#### 5.7.4.5 int32\_t TFlow::DNSIndex

points to DNS-name in a STRING-table, -1 = no DNS

#### 5.7.4.6 int32\_t TFlow::HTTPIndex

if HTTP-flow, this index points to a special object that identifies triggers of new flows, 0 = no HTTP-traffic

#### 5.7.4.7 uint16\_t TFlow::Identification

Identification field (only defined for DNS, 0 for other protocols)

#### 5.7.4.8 uint32\_t TFlow::LocalIP

IP-address of the observed local computer.

#### 5.7.4.9 uint16\_t TFlow::LocalPort

UDP/TCP=Port, ICMP=identifier, Other=0.

#### 5.7.4.10 uint32\_t TFlow::LocalSEQ

Local SEQ (0 if undefined)

#### 5.7.4.11 int32\_t TFlow::NextFlowIndex

Links to a next flow by an index (a kind of linked list) -1 = no next flow.

#### 5.7.4.12 int32\_t TFlow::NumberOfReceivedBytes

Aggregated number of transmitted bytes.

**5.7.4.13 int32\_t TFlow::NumberOfReceivedPackets**

Aggregated number of transmitted bytes.

**5.7.4.14 int32\_t TFlow::NumberOfTransmittedBytes**

Aggregated number of transmitted bytes.

**5.7.4.15 int32\_t TFlow::NumberOfTransmittedPackets**

Aggregated number of transmitted bytes.

**5.7.4.16 int32\_t TFlow::ParentFlow**

id of parent flow in the tree, 0=no parents (root flow)

**5.7.4.17 uint8\_t TFlow::Protocol**

1=ICMP, 6=TCP, 17=UDP, 255=IP-other

**5.7.4.18 uint32\_t TFlow::RemoteIP**

IP-address of the remote computer.

**5.7.4.19 uint16\_t TFlow::RemotePort**

UDP/TCP=Port ICMP=sequence number, Other=0.

**5.7.4.20 uint32\_t TFlow::RemoteSEQ**

Remote expected SEQ (0 if undefined)

**5.7.4.21 uint8\_t TFlow::Resolver**

bit 0..3=resolver protocol (0=nothing, 1=DNS, 2..15=spare, bit 7=1=resolved)

**5.7.4.22 int64\_t TFlow::StartTime**

Time of first packet in Flow in useconds since 1970.

#### 5.7.4.23 `uint8_t TFlow::Status`

Status is the current status of the flow: 0=empty, 1=aggregated from 1 packet, 2=aggregated from more packets.

#### 5.7.4.24 `int64_t TFlow::StopTime`

Time of last packet in Flow in useconds since 1970.

#### 5.7.4.25 `uint8_t TFlow::TCPFlag`

Aggregated TCP-Flags if TCP-Flow.

#### 5.7.4.26 `int32_t TFlow::TreeIndex`

id of the flow tree (tree id)

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

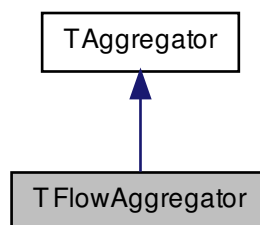
- [src/Flow.h](#)
- [src/Flow.cpp](#)

## 5.8 TFlowAggregator Class Reference

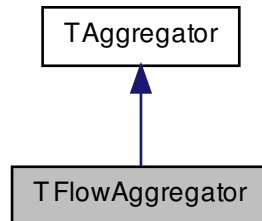
Important class that manages the flows.

```
#include <FlowAggregator.h>
```

Inheritance diagram for TFlowAggregator:



Collaboration diagram for TFlowAggregator:



### Public Member Functions

- [TFlowAggregator](#) (uint32\_t lIP)
- uint8\_t [add](#) (void)
- void [dump](#) (char \*content, int dest)

### Public Attributes

- uint8\_t [Direction](#)  
*Flow direction of last analyzed packet: INGRESS, EGRESS or UNDEFINED.*

### Private Member Functions

- uint8\_t [addPacket](#) (void)
- uint8\_t [addFlow](#) (void)
- uint8\_t [deleteFlow](#) (int32\_t i)
- uint8\_t [find](#) (void)
- uint8\_t [find](#) (uint8\_t p, uint32\_t sIP, uint32\_t dIP, uint16\_t sP, uint16\_t dP, uint16\_t id)

### Private Attributes

- uint32\_t [LocalIP](#)  
*IP-address of local as a reference for the flow direction.*
- int32\_t [TotalPacketCounter](#)
- int32\_t [InPacketCounter](#)
- int32\_t [OutPacketCounter](#)

- `int32_t` [NoLocalPacketCounter](#)
- `int32_t` [EgressFlowCounter](#)
- `int32_t` [IngressFlowCounter](#)
- `int32_t` [AlreadyOpenCounter](#)
- `int32_t` [EgressUDPFlowCounter](#)
- `int32_t` [NonEmptyRRUDPCounter](#)
- `int32_t` [IngressUDPFlowCounter](#)
- `int32_t` [EgressTCPFlowCounter](#)
- `int32_t` [IngressTCPFlowCounter](#)
- `int32_t` [EgressICMPFlowCounter](#)
- `int32_t` [IngressICMPFlowCounter](#)
- `int32_t` [EgressOtherFlowCounter](#)
- `int32_t` [IngressOtherFlowCounter](#)
- `int32_t` [EgressHTTPFlowCounter](#)
- `int32_t` [EgressHTTPSFlowCounter](#)
- `int32_t` [EgressDNSFlowCounter](#)
- `int64_t` [StartTime](#)

### 5.8.1 Detailed Description

Important class that manages the flows.

< It creates a static array of FLOWBUFFERSIZE (65536) Flow-objects. By inspection of packets it creates or updates flows. For fast search a hash table is used with FLOWHASHSIZE (1024) entries. Each entry is the start of a linked list of Flows. Instead of Pointers the array-indices are used in the linked list. New flows are placed in a randomly picked empty flow of the static array.

### 5.8.2 Constructor & Destructor Documentation

#### 5.8.2.1 TFlowAggregator::TFlowAggregator ( uint32\_t *IP* )

Constructor. During creation a fixed array of Flows with hashtable is created for storage.

##### Parameters

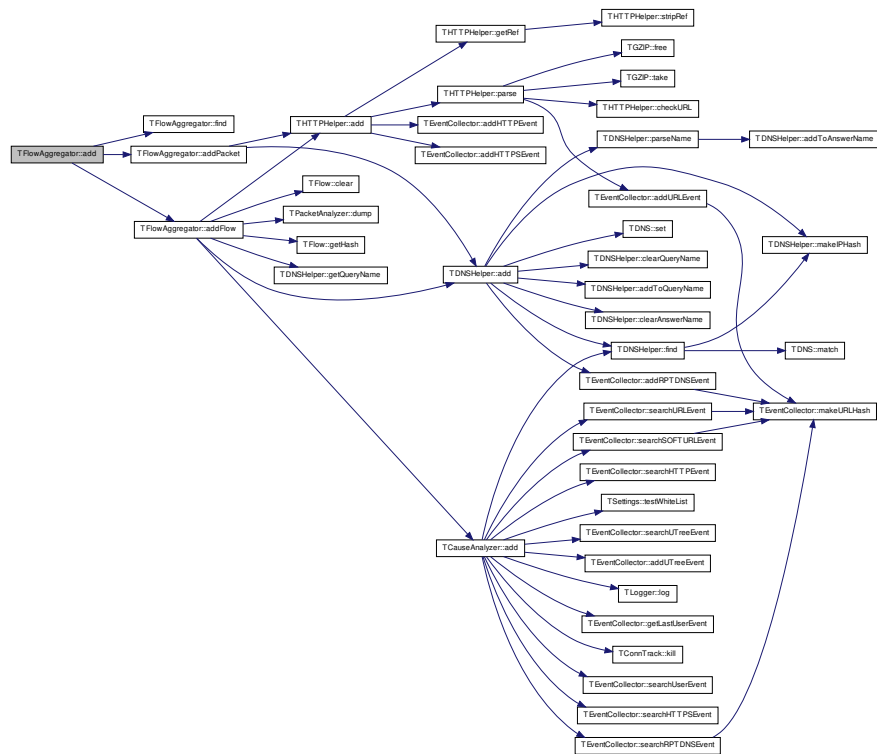
<i>IP</i>	The IP-address of the observed device. This will be the local IP-address.
-----------	---

### 5.8.3 Member Function Documentation

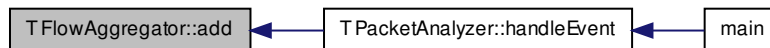
#### 5.8.3.1 uint8\_t TFlowAggregator::add ( void ) [virtual]

Reimplemented from [TAggregator](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.8.3.2 uint8\_t TFlowAggregator::addFlow ( void ) [private]

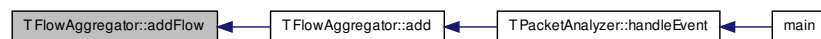
Makes a new flow of the current Packet in PacketAnalyzer

1=ok, 0=Buffer full or no matching LocalIP or No SYN in HTTP(S)

Here is the call graph for this function:



Here is the caller graph for this function:





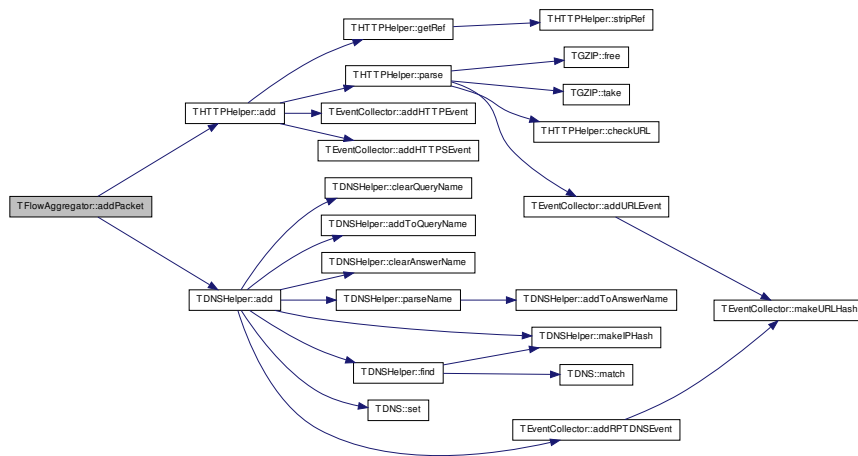
### 5.8.3.3 uint8\_t TFlowAggregator::addPacket ( void ) [private]

adds packet to existing flow or creates a new flow. Important to call first "find", because it uses Index to link with an existing flow.

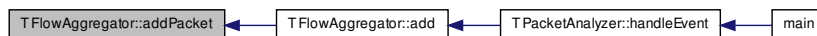
#### Returns

1=success

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.8.3.4 uint8\_t TFlowAggregator::deleteFlow ( int32\_t i ) [private]

Frees a flow. Untested at this moment.

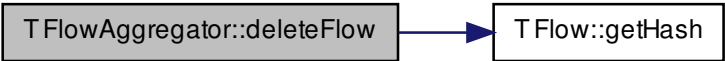
#### Returns

1=ok

#### Parameters

<i>i</i>	Index of the flow in the static array of flows.
----------	---

Here is the call graph for this function:



5.8.3.5 void TFlowAggregator::dump ( char \* *content*, int *dest* )

updates existing flow or creates a new flow with the current packet in PacketAnalyzer.  
return 1 = success Dumps the content of all flows to stdout or logfile

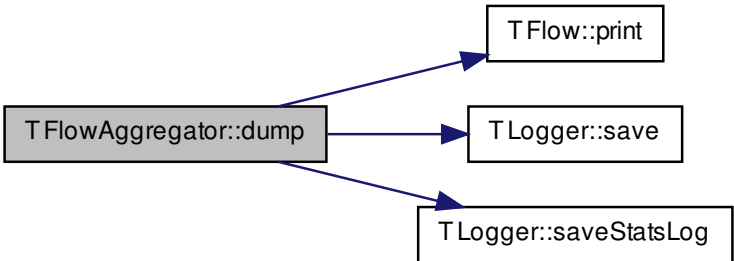
Returns

void

Parameters

<i>*content</i>	pointer to content that must be printed. NULL is print to stdout
<i>dest</i>	destination of the log: 0= to stdout, 1=to logfile

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.8.3.6 `uint8_t TFlowAggregator::find ( void ) [private]`

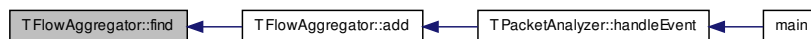
Finds the flow of current packet in PacketAnalyzer if it exists.

##### Returns

result of the find action

- 1 = Flow found, the index can be found in Index
- 2 = Flow not found, Index=-1
- 0 = No local IP present in the Packet

Here is the caller graph for this function:



#### 5.8.3.7 `uint8_t TFlowAggregator::find ( uint8_t p, uint32_t sIP, uint32_t dIP, uint16_t sP, uint16_t dP, uint16_t id ) [private]`

Finds the flow specified IP/Port parameters. return 1=success

##### Returns

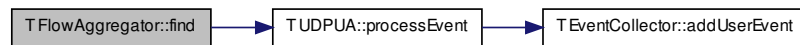
result of the find action

- 1 = Flow found, the index can be found in Index
- 2 = Flow not found, Index=-1
- 0 = No local IP present in the Packet

## Parameters

<i>p</i>	Protocol number (e.g. 6=TCP)
<i>sIP</i>	Source IP address
<i>dIP</i>	Destination IP address
<i>sP</i>	Source Port
<i>dP</i>	Destination Port
<i>id</i>	Identification field, as used in DNS, 0 for other protocols

Here is the call graph for this function:



#### 5.8.4 Member Data Documentation

5.8.4.1 `int32_t TFlowAggregator::AlreadyOpenCounter` [private]

5.8.4.2 `uint8_t TFlowAggregator::Direction`

Flow direction of last analyzed packet: INGRESS, EGRESS or UNDEFINED.

5.8.4.3 `int32_t TFlowAggregator::EgressDNSFlowCounter` [private]

5.8.4.4 `int32_t TFlowAggregator::EgressFlowCounter` [private]

5.8.4.5 `int32_t TFlowAggregator::EgressHTTPFlowCounter` [private]

5.8.4.6 `int32_t TFlowAggregator::EgressHTTPSFlowCounter` [private]

5.8.4.7 `int32_t TFlowAggregator::EgressICMPFlowCounter` [private]

5.8.4.8 `int32_t TFlowAggregator::EgressOtherFlowCounter` [private]

5.8.4.9 `int32_t TFlowAggregator::EgressTCPFlowCounter` [private]

5.8.4.10 `int32_t TFlowAggregator::EgressUDPFlowCounter` [private]

5.8.4.11 `int32_t TFlowAggregator::IngressFlowCounter` [private]

5.8.4.12 `int32_t TFlowAggregator::IngressICMPFlowCounter` [private]

5.8.4.13 `int32_t TFlowAggregator::IngressOtherFlowCounter` [private]

5.8.4.14 `int32_t TFlowAggregator::IngressTCPFlowCounter` [private]

5.8.4.15 `int32_t TFlowAggregator::IngressUDPFlowCounter` [private]

5.8.4.16 `int32_t TFlowAggregator::InPacketCounter` [private]

5.8.4.17 `uint32_t TFlowAggregator::LocalIP` [private]

IP-address of local as a reference for the flow direction.

5.8.4.18 `int32_t TFlowAggregator::NoLocalPacketCounter` [private]

5.8.4.19 `int32_t TFlowAggregator::NonEmptyRRUDPCounter` [private]

5.8.4.20 `int32_t TFlowAggregator::OutPacketCounter` [private]

5.8.4.21 `int64_t TFlowAggregator::StartTime` [private]

5.8.4.22 `int32_t TFlowAggregator::TotalPacketCounter` [private]

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

- [src/FlowAggregator.h](#)
- [src/FlowAggregator.cpp](#)

## 5.9 TGZIP Class Reference

Class for deflating gzip data. Wraps around zlib.

```
#include <GZIP.h>
```

### Public Member Functions

- [TGZIP](#) (void)
- `int isInUse` (void)
- `void take` (void)
- `void free` (void)
- `int uncompress` (void)

### Public Attributes

- `int Processing`
- `z_stream strm`
- `unsigned char * InBuffer`
- `int InLength`

- unsigned char [OutBuffer](#) [GZIP\_BUFFERSIZE]
- int [OutLength](#)

### Private Attributes

- int [InUse](#)

## 5.9.1 Detailed Description

Class for deflating gzip data. Wraps around zlib.

<

## 5.9.2 Constructor & Destructor Documentation

### 5.9.2.1 TGZIP::TGZIP ( void )

The constructor. Resets all the data-members, including NextFlow

Here is the call graph for this function:



## 5.9.3 Member Function Documentation

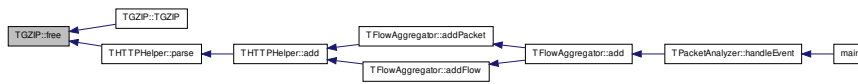
### 5.9.3.1 void TGZIP::free ( void )

Frees the gzip. Memory is flushed cleared etc.

**Returns**

void

Here is the caller graph for this function:

**5.9.3.2 int TGZIP::isInUse ( void )**

Tests if the object is free to take.

**Returns**

0=free 1=in use

**5.9.3.3 void TGZIP::take ( void )**

Takes the gzip-object. Memory is assumed to be flushed, cleared

**Returns**

void

Here is the caller graph for this function:

**5.9.3.4 int TGZIP::uncompress ( void )**

Uncompresses a blok of data. Inlength and \*Inbuffer must be prepared before this call

**Returns**

int 0=ok, -1 wrong magic number, -2=no data left

### 5.9.4 Member Data Documentation

#### 5.9.4.1 unsigned char\* TGZIP::InBuffer

Pointer to the input buffer

#### 5.9.4.2 int TGZIP::InLength

Lenth of the input buffer

#### 5.9.4.3 int TGZIP::InUse [private]

Boolean 0=object is free, 1=object is already taken

#### 5.9.4.4 unsigned char TGZIP::OutBuffer[GZIP\_BUFFERSIZE]

Pointer to the output buffer

#### 5.9.4.5 int TGZIP::OutLength

Lenth of the output buffer

#### 5.9.4.6 int TGZIP::Processing

#### 5.9.4.7 z\_stream TGZIP::strm

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

- [src/GZIP.h](#)
- [src/GZIP.cpp](#)

## 5.10 THTTP Class Reference

Storage class for HTTP-events. Every HTTP-flow or HTTPS-flow that can potentially trigger new traffi by embedded URL's has an object of this class.

```
#include <HTTP.h>
```

### Public Member Functions

- [THTTP](#) (void)
- void [clear](#) (void)
- void [print](#) (char \*content)



## Public Attributes

- int64\_t [Time](#)  
*Timestamp of the last HTTP-event.*
- uint8\_t [Status](#)  
*State of the HTTP-dialog.*
- uint32\_t [InByteCounter](#)  
*Number of ingress bytes during a receive phase.*
- uint32\_t [OutByteCounter](#)  
*Number of egress bytes during a send phase.*
- uint32\_t [TotalInByteCounter](#)  
*Total number of egress payload bytes.*
- uint32\_t [TotalOutByteCounter](#)  
*Total number of ingress payload bytes.*
- int32\_t [FlowIndex](#)  
*Index to Flow that created the cause.*
- uint8\_t [ParseState](#)  
*Main State of the FSM for parsing URL's.*
- uint8\_t [ParseSubState](#)  
*Sub State inside a Main State for parsing URL's.*
- uint16\_t [ParseMicroState](#)  
*Micro State inside a SubState for parsing URL's.*
- int32\_t [PayloadSize](#)  
*Size of the HTTP(S) payload in the last received packet.*
- uint8\_t [ContentType](#)  
*0=unknown, 1=text*
- uint8\_t [Encoding](#)  
*0=unknown, 1=gzip*
- uint8\_t [Chunked](#)  
*0=unknown, 1=chunked*
- int32\_t [GZIPIndex](#)  
*pointer to GZIP-object if Encoding=1*
- int [PostDotLength](#)
- uint8\_t [RefStat](#)  
*Status of Referrer field (0=waitig , 1= 1=first parse failed, 2=second parse failed).*
- char [URLBuffer](#) [256]  
*Buffer to hold URL if it passes over multiple packets.*
- char [RefBuffer](#) [80]  
*Buffer to hold Referer.*
- int64\_t [LastHeaderTime](#)
- int64\_t [LastTailTime](#)

### 5.10.1 Detailed Description

Storage class for HTTP-events. Every HTTP-flow or HTTPS-flow that can potentially trigger new traffi by embedded URL's has an object of this class.

<

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 THTTP::THTTP ( void )

The constructor. Resets all the data-members, including NextFlow

Here is the call graph for this function:



### 5.10.3 Member Function Documentation

#### 5.10.3.1 void THTTP::clear ( void )

Clears all the data-members.

##### Returns

void

Here is the caller graph for this function:



### 5.10.3.2 void THTTP::print ( char \* *content* )

Prints a cause in readable format to stdout

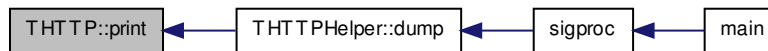
Returns

void

Parameters

<i>*content</i>	pointer to content that must be printed. NULL is print to stdout
-----------------	--

Here is the caller graph for this function:



## 5.10.4 Member Data Documentation

### 5.10.4.1 uint8\_t THTTP::Chunked

0=unknown, 1=chunked

### 5.10.4.2 uint8\_t THTTP::ContentType

0=unknown, 1=text

### 5.10.4.3 uint8\_t THTTP::Encoding

0=unknown, 1=gzip

### 5.10.4.4 int32\_t THTTP::FlowIndex

Index to Flow that created the cause.

### 5.10.4.5 int32\_t THTTP::GZIPIndex

pointer to GZIP-object if Encoding=1

**5.10.4.6 uint32\_t THTTP::InByteCounter**

Number of ingress bytes during a receive phase.

**5.10.4.7 int64\_t THTTP::LastHeaderTime****5.10.4.8 int64\_t THTTP::LastTailTime****5.10.4.9 uint32\_t THTTP::OutByteCounter**

Number of egress bytes during a send phase.

**5.10.4.10 uint16\_t THTTP::ParseMicroState**

Micro State inside a SubState for parsing URL's.

**5.10.4.11 uint8\_t THTTP::ParseState**

Main State of the FSM for parsing URL's.

**5.10.4.12 uint8\_t THTTP::ParseSubState**

Sub State inside a Main State for parsing URL's.

**5.10.4.13 int32\_t THTTP::PayloadSize**

Size of the HTTP(S) payload in the last received packet.

**5.10.4.14 int THTTP::PostDotLength**

-1=no dot detected, other is number of characters after the dot

**5.10.4.15 char THTTP::RefBuffer[80]**

Buffer to hold Referer.

**5.10.4.16 uint8\_t THTTP::RefStat**

Status of Referrer field (0=waiting, 1=1=first parse failed, 2=second parse failed).

**5.10.4.17 uint8\_t THTTP::Status**

State of the HTTP-dialog.

#### 5.10.4.18 int64\_t THTTP::Time

Timestamp of the last HTTP-event.

#### 5.10.4.19 uint32\_t THTTP::TotalInByteCounter

Total number of egress payload bytes.

#### 5.10.4.20 uint32\_t THTTP::TotalOutByteCounter

Total number of ingress payload bytes.

#### 5.10.4.21 char THTTP::URLBuffer[256]

Buffer to hold URL if it passes over multiple packets.

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

- [src/HTTP.h](#)
- [src/HTTP.cpp](#)

## 5.11 THTTPEvent Class Reference

```
#include <EventCollector.h>
```

### Public Member Functions

- [THTTPEvent](#) (void)  
*The constructor. Clears all the data-members.*
- void [clear](#) (void)  
*Clears all data-members.*
- void [print](#) (char \*content)  
*Prints a cause in readable format to stdout.*

### Public Attributes

- int64\_t [TimeStamp](#)  
*TimeStamp: Time since 1970 in us.*
- uint8\_t [EventCode](#)  
*EventNumber: 0=UNDEFINED, 1=HTTPDATA RECEIVED, 2=HTTPDATA PUSH, 3=-CAUSE\_HTTPWITHDRAWN.*
- int32\_t [FlowIndex](#)  
*Index to flow that contains the event.*

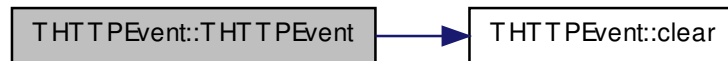
- uint8\_t [CauseCount](#)  
*number of times the event caused new flows*
- int64\_t [DeltaT](#)  
*Time contributed to openwindow.*

### 5.11.1 Constructor & Destructor Documentation

#### 5.11.1.1 THTTPEvent::THTTPEvent ( void )

The constructor. Clears all the data-members.

Here is the call graph for this function:



### 5.11.2 Member Function Documentation

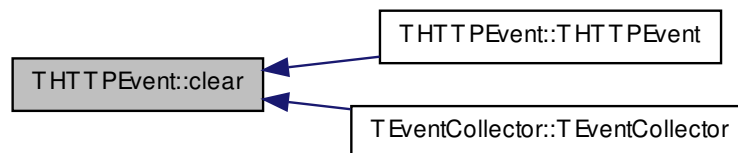
#### 5.11.2.1 void THTTPEvent::clear ( void )

Clears all data-members.

Returns

void

Here is the caller graph for this function:



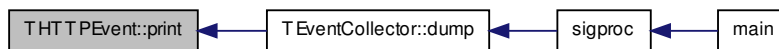
### 5.11.2.2 void THTTPEvent::print ( char \* *content* )

Prints a cause in readable format to stdout.

#### Returns

void

Here is the caller graph for this function:



### 5.11.3 Member Data Documentation

#### 5.11.3.1 uint8\_t THTTPEvent::CauseCount

number of times the event caused new flows

#### 5.11.3.2 int64\_t THTTPEvent::DeltaT

Time contributed to openwindow.

#### 5.11.3.3 uint8\_t THTTPEvent::EventCode

EventNumber: 0=UNDEFINED, 1=HTTPDATARECEIVED, 2=HTTPDATAPUSH, 3=CAUSE\_HTTPWITHDRAWN.

#### 5.11.3.4 int32\_t THTTPEvent::FlowIndex

Index to flow that contains the event.

#### 5.11.3.5 int64\_t THTTPEvent::TimeStamp

TimeStamp: Time since 1970 in us.

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

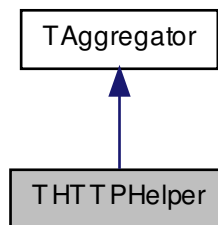
- [src/EventCollector.h](#)
- [src/EventCollector.cpp](#)

## 5.12 THTTPHelper Class Reference

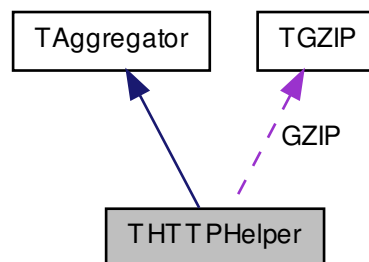
Class that aggregates HTTP traffic. It aggregates information that predicts the cause of new traffic flows. [THTTPHelper](#) creates a static array of HTTP\_BUFFER\_SIZE (65536) HTTP-objects. The ClusterAggregator uses this class to check for potential HTTP-parent flows. [TFlowAggregator](#) updates the information of this class.

```
#include <HTTPHelper.h>
```

Inheritance diagram for THTTPHelper:



Collaboration diagram for THTTPHelper:



### Public Member Functions

- [THTTPHelper](#) (void)
- `uint8_t add` (void)



- void [dump](#) (char \*dm, int dest)

### Private Member Functions

- int [parse](#) (void)
- int [getRef](#) (void)
- int [stripRef](#) (void)
- int [checkURL](#) (void)

### Private Attributes

- [TGZIP GZIP](#) [[TOTAL\\_GZIP](#)]  
*GZIP-objects.*
- char [DeChunkedContent](#) [32768]  
*intermediate buffer for chunk header removal*
- uint32\_t [SuccessRefCounter](#)
- uint32\_t [RefCounter](#)
- uint32\_t [GetRequestCounter](#)
- int32\_t [GZIPIndex](#)  
*index to most recent GZIP-buffer or -1 if empty*

#### 5.12.1 Detailed Description

Class that aggregates HTTP traffic. It aggregates information that predicts the cause of new traffic flows. [THTTPHelper](#) creates a static array of HTTP\_BUFFER\_SIZE (65536) HTTP-objects. The ClusterAggregator uses this class to check for potential HTTP-parent flows. [TFlowAggregator](#) updates the information of this class.

<

#### 5.12.2 Constructor & Destructor Documentation

##### 5.12.2.1 THTTPHelper::THTTPHelper ( void )

Constructor. During creation a fixed array of Causes is created for storage.

#### 5.12.3 Member Function Documentation

##### 5.12.3.1 uint8\_t THTTPHelper::add ( void ) [virtual]

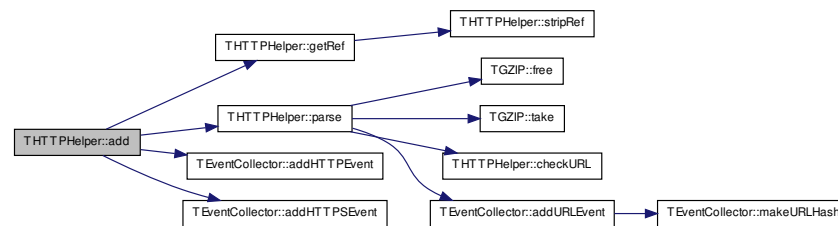
Adds HTTP-packets of new or existing flows to the aggregator

**Returns**

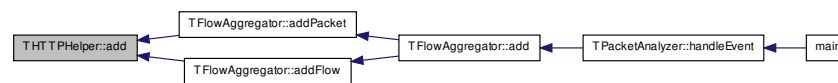
1=success: data updated in existing records, 2=success: new HTTP-record added, 0 = parse error, 10=Cannot create because HTTPBuffer Full

Reimplemented from [TAggregator](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.12.3.2 `int THTTPHelper::checkURL ( void ) [private]`

Here is the caller graph for this function:



### 5.12.3.3 `void THTTPHelper::dump ( char * dm, int dest )`

Dumps the content of all recent causes to stdout

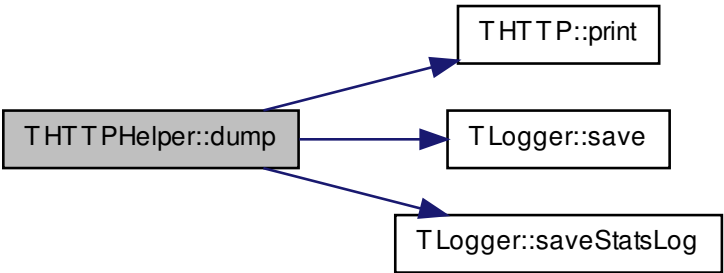
**Returns**

void

Parameters

<i>*dm</i>	pointer to content that must be printed. NULL is print to stdout
<i>dest</i>	destination of the log: 0= to stdout, 1=to logfile

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.3.4 `int THTTPHelper::getRef ( void ) [private]`

Parses HTTP get requests for referred domains.

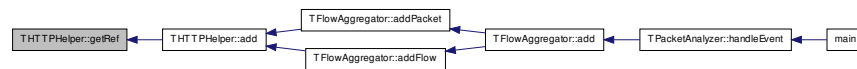
**Returns**

1 = referred domain found, 2 = referred domain parsed but not found, 0 = no referred domain parsed, -1 = error

Here is the call graph for this function:



Here is the caller graph for this function:

**5.12.3.5** `int THTTPHelper::parse ( void )` [`private`]

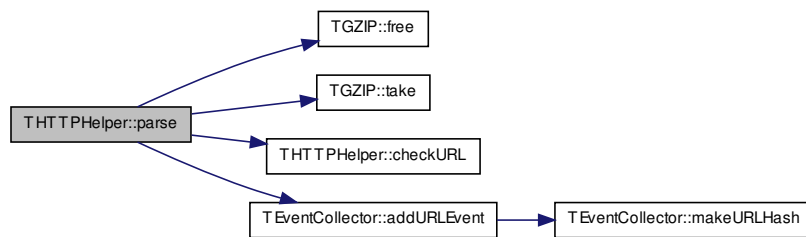
Parses HTTP. First the header and subsequently the body if the content-type=text. - Dechunking and decompression is included if necessary. The domain names of all url's, found in the body, are placed in the URL\_EventList.

**Returns**

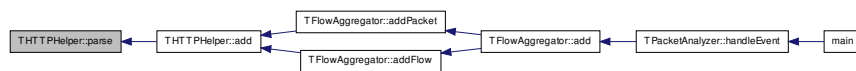
0 or positive=number of parsed URL's, negative=failure

Start first gzip

Here is the call graph for this function:



Here is the caller graph for this function:



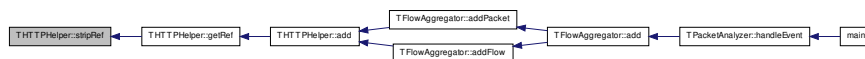
### 5.12.3.6 int THTTPHelper::stripRef ( void ) [private]

Strips the domain part of the referer from other stuff.

**Returns**

1 = referred domain successfully stripped

Here is the caller graph for this function:



## 5.12.4 Member Data Documentation

#### 5.12.4.1 `char THTTPHelper::DeChunkedContent[32768]` [private]

intermediate buffer for chunk header removal

#### 5.12.4.2 `uint32_t THTTPHelper::GetRequestCounter` [private]

#### 5.12.4.3 `TGZIP THTTPHelper::GZIP[TOTAL_GZIP]` [private]

GZIP-objects.

#### 5.12.4.4 `int32_t THTTPHelper::GZIPIndex` [private]

index to most recent GZIP-buffer or -1 if empty

#### 5.12.4.5 `uint32_t THTTPHelper::RefCounter` [private]

#### 5.12.4.6 `uint32_t THTTPHelper::SuccessRefCounter` [private]

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

- [src/HTTPHelper.h](#)
- [src/HTTPHelper.cpp](#)

## 5.13 THTTPSEvent Class Reference

```
#include <EventCollector.h>
```

### Public Member Functions

- [THTTPSEvent](#) (void)  
*The constructor. Clears all the data-members.*
- void [clear](#) (void)  
*Clears all data-members.*
- void [print](#) (char \*content)  
*Prints a cause in readable format to stdout.*

### Public Attributes

- int64\_t [TimeStamp](#)  
*TimeStamp: Time since 1970 in us.*
- uint8\_t [EventCode](#)

*EventNumber: 0=UNDEFINED, 1=HTTPDATA RECEIVED, 2=HTTPDATA PUSH, 3=CAUSE\_HTTPWITHDRAWN.*

- int32\_t [FlowIndex](#)

*Index to flow that contains the event.*

- uint8\_t [CauseCount](#)

*number of times the event caused new flows*

- int64\_t [DeltaT](#)

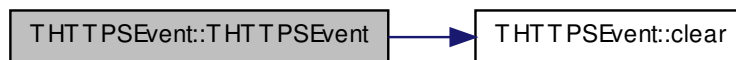
*Time contributed to openwindow.*

### 5.13.1 Constructor & Destructor Documentation

#### 5.13.1.1 THTTPSEvent::THTTPSEvent ( void )

The constructor. Clears all the data-members.

Here is the call graph for this function:



### 5.13.2 Member Function Documentation

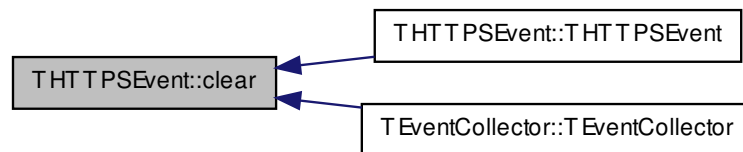
#### 5.13.2.1 void THTTPSEvent::clear ( void )

Clears all data-members.

**Returns**

void

Here is the caller graph for this function:

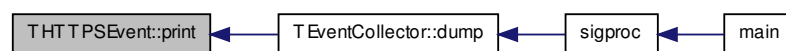
**5.13.2.2 void THTTPSEvent::print ( char \* *content* )**

Prints a cause in readable format to stdout.

**Returns**

void

Here is the caller graph for this function:

**5.13.3 Member Data Documentation****5.13.3.1 uint8\_t THTTPSEvent::CauseCount**

number of times the event caused new flows

**5.13.3.2 int64\_t THTTPSEvent::DeltaT**

Time contributed to openwindow.



#### 5.13.3.3 uint8\_t THTTPSEvent::EventCode

EventNumber: 0=UNDEFINED, 1=HTTPDATA RECEIVED, 2=HTTPDATA PUSH, 3=CAUSE\_HTTPWITHDRAWN.

#### 5.13.3.4 int32\_t THTTPSEvent::FlowIndex

Index to flow that contains the event.

#### 5.13.3.5 int64\_t THTTPSEvent::TimeStamp

TimeStamp: Time since 1970 in us.

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

- [src/EventCollector.h](#)
- [src/EventCollector.cpp](#)

## 5.14 TLogger Class Reference

Class that aggregates log messages in flat file format in RAM.

```
#include <Logger.h>
```

### Public Member Functions

- [TLogger](#) (const char \*name)
- int [log](#) (int sev, int fac, char \*message)
- int [log](#) (int sev, int fac, int32\_t flowindex, char \*id, int64\_t delay)
- int [saveLog](#) (void)
- int [save](#) (const char \*ext, char \*dm)
- int [initStatsLog](#) (void)
- int [saveStatsLog](#) (char \*dm)

### Private Attributes

- int [LineCounter](#)
- char [Log](#) [LOGSIZE]
- char \* [Buffer](#)
- const char \* [FileName](#)

### 5.14.1 Detailed Description

Class that aggregates log messages in flat file format in RAM.

< Finally it can save the log in a file. Every field is separated with a comma. Every record is separated with a newline. The format is:

Type1: Timestamp, Severity, Facility, message

Type2: Timestamp, Severity, Facility, message, ID, delay

Type3: Timestamp, Severity, Facility, message, index

### 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 TLogger::TLogger ( const char \* *name* )

Constructor. During creation a fixed array of Causes is created for storage.

##### Returns

void

##### Parameters

<i>*name</i>	filename
--------------	----------

### 5.14.3 Member Function Documentation

#### 5.14.3.1 int TLogger::initStatsLog ( void )

Initializes a new file for statistics

##### Returns

0=success, -1=error

Here is the caller graph for this function:



### 5.14.3.2 int TLogger::log ( int *sev*, int *fac*, char \* *message* )

Writes a generic log record

#### Returns

0=success, -1=error

#### Parameters

<i>sev</i>	Severity (0 t/m 7)
<i>fac</i>	Facility (number identifying which functional unit is the source)
* <i>message</i>	string with the message to write

Here is the caller graph for this function:



### 5.14.3.3 int TLogger::log ( int *sev*, int *fac*, int32\_t *flowindex*, char \* *id*, int64\_t *delay* )

Writes a new flow record

#### Returns

0=success, -1=error

#### Parameters

<i>sev</i>	Severity (0 t/m 7)
<i>fac</i>	Facility (number identifying which functional unit is the source)
<i>flowindex</i>	Index of the new flow
* <i>id</i>	IP or name of the new flow
<i>delay</i>	Time in us between new flow and cause

### 5.14.3.4 int TLogger::save ( const char \* *ext*, char \* *dm* )

Saves a string in a file with specified extension

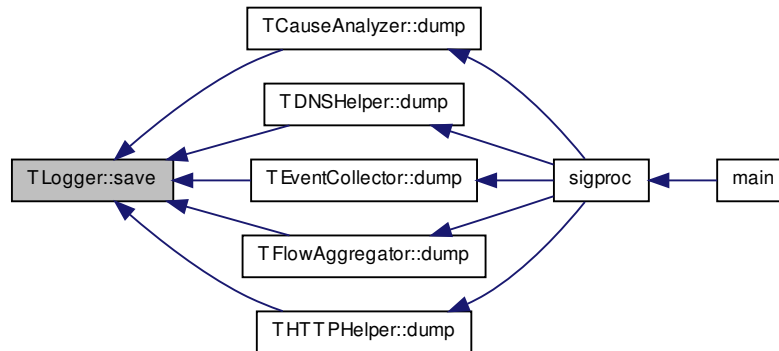
#### Returns

0=success, -1=error

## Parameters

<i>*ext</i>	pointer to extension of the filename, starting with a dot e.g. .flow, .dat
<i>*dm</i>	pointer to complete string that must be saved

Here is the caller graph for this function:



#### 5.14.3.5 int TLogger::saveLog ( void )

Saves the complete log in a file

## Returns

0=success, -1=error

Here is the caller graph for this function:



#### 5.14.3.6 int TLogger::saveStatsLog ( char \* dm )

Adds a string with statistics to the statistics file

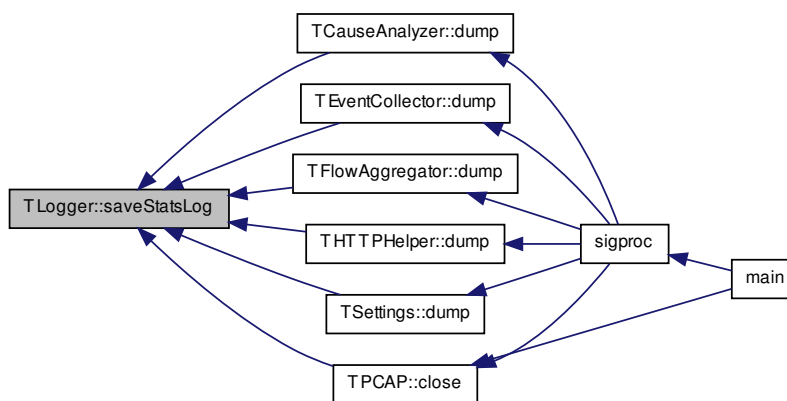
**Returns**

0=success, -1=error

**Parameters**

<i>*dm</i>	pointer to complete string that must be saved
------------	---

Here is the caller graph for this function:

**5.14.4 Member Data Documentation****5.14.4.1** `char* TLogger::Buffer` [private]

Pointer in the storage

**5.14.4.2** `const char* TLogger::FileName` [private]

File name after save to non-volatile memory

**5.14.4.3** `int TLogger::LineCounter` [private]

Number of lines in the log

**5.14.4.4** `char TLogger::Log[LOGSIZE]` [private]

RAM Storage for logging

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

- src/[Logger.h](#)
- src/[Logger.cpp](#)

## 5.15 TPacketAnalyzer Class Reference

Class for analyzing an arrived packet.

```
#include <PacketAnalyzer.h>
```

### Public Member Functions

- [TPacketAnalyzer](#) ()
- void [handleEvent](#) (void)
- void [dump](#) (void)

### Public Attributes

- int64\_t [Time](#)  
*Time of Packet-arrival in seconds sinze 1970.*
- int32\_t [Length](#)  
*Number of available packet-bytes.*
- uint8\_t [Protocol](#)  
*IP-Protocol-Field:1=ICMP, 6=TCP, 17=UDP, 255=IP-other.*
- uint32\_t [SourceIP](#)  
*IP source sddress.*
- uint32\_t [DestIP](#)  
*IP destination address.*
- uint16\_t [SourcePort](#)  
*PORT if UDP/TCP else IDENTIFIER if ICMP or 0 if other L4-protocol.*
- uint16\_t [DestPort](#)  
*PORT if UDP/TCP else SEQUENCE NUMBER if ICMP or 0 if other L4-protocol.*
- uint8\_t [TCPFlag](#)  
*TCP FlagRegister bit0=FIN, bit1=SYN, bit2=RST, bit3=PSH bit4=ACK, bit 5=URG.*
- uint32\_t [TCPSEQ](#)  
*TCP SEQ NR.*
- uint32\_t [TCPACK](#)  
*TCP ACK NR.*
- uint16\_t [Identification](#)  
*Identification (DNS)*
- int32\_t [PayloadIndex](#)  
*Start of the Payload after L4 in Packet.*
- int8\_t [TCPValid](#)  
*1=TCP\_Packet expected, 0=Undefined, -1=TCP\_Packet unexpected (written from - Flow)*

## Private Member Functions

- `int8_t makePacket (void)`

### 5.15.1 Detailed Description

Class for analyzing an arrived packet.

< If a packet arrives, this singleton will handle the event.

After analysis of L3 and L4, the FlowAnalyzer class is called for further processing.

### 5.15.2 Constructor & Destructor Documentation

#### 5.15.2.1 TPacketAnalyzer::TPacketAnalyzer ( )

The constructor. Resets all the data-members

### 5.15.3 Member Function Documentation

#### 5.15.3.1 void TPacketAnalyzer::dump ( void )

Dumps packet shortlist to stdout for debug purpose.

#### Returns

void

Here is the caller graph for this function:

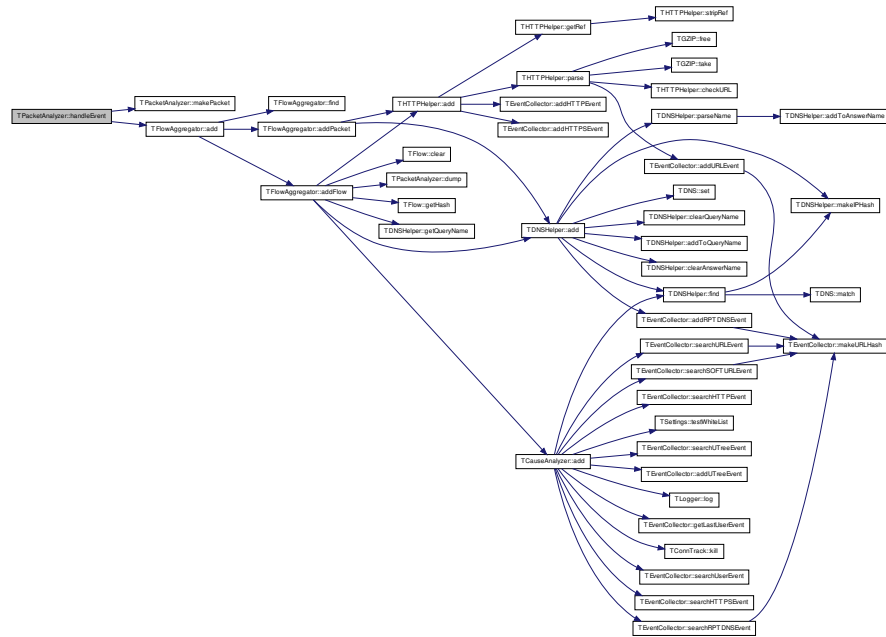


#### 5.15.3.2 void TPacketAnalyzer::handleEvent ( void )

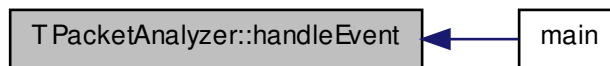
Analyzes a packet and fills the public members with actual values of the arrived packet.

void

Here is the call graph for this function:



Here is the caller graph for this function:





### 5.15.3.3 int8\_t TPacketAnalyzer::makePacket ( void ) [private]

Here is the caller graph for this function:



## 5.15.4 Member Data Documentation

### 5.15.4.1 uint32\_t TPacketAnalyzer::DestIP

IP destination address.

### 5.15.4.2 uint16\_t TPacketAnalyzer::DestPort

PORT if UDP/TCP else SEQUENCE NUMBER if ICMP or 0 if other L4-protocol.

### 5.15.4.3 uint16\_t TPacketAnalyzer::Identification

Identification (DNS)

### 5.15.4.4 int32\_t TPacketAnalyzer::Length

Number of available packet-bytes.

### 5.15.4.5 int32\_t TPacketAnalyzer::PayloadIndex

Start of the Payload after L4 in Packet.

### 5.15.4.6 uint8\_t TPacketAnalyzer::Protocol

IP-Protocol-Field:1=ICMP, 6=TCP, 17=UDP, 255=IP-other.

### 5.15.4.7 uint32\_t TPacketAnalyzer::SourceIP

IP source address.

#### 5.15.4.8 `uint16_t TPacketAnalyzer::SourcePort`

PORT if UDP/TCP else IDENTIFIER if ICMP or 0 if other L4-protocol.

#### 5.15.4.9 `uint32_t TPacketAnalyzer::TCPACK`

TCP ACK NR.

#### 5.15.4.10 `uint8_t TPacketAnalyzer::TCPFlag`

TCP FlagRegister bit0=FIN, bit1=SYN, bit2=RST, bit3=PSH bit4=ACK, bit 5=URG.

#### 5.15.4.11 `uint32_t TPacketAnalyzer::TCPSEQ`

TCP SEQ NR.

#### 5.15.4.12 `int8_t TPacketAnalyzer::TCPValid`

1=TCP\_Packet expected, 0=Undefined, -1=TCP\_Packet unexpected (written from Flow)

#### 5.15.4.13 `int64_t TPacketAnalyzer::Time`

Time of Packet-arrival in seconds sinze 1970.

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

- [src/TPacketAnalyzer.h](#)
- [src/TPacketAnalyzer.cpp](#)

## 5.16 TPCAP Class Reference

The "singleton" object of [TPCAP](#) is a wrapper around the libpcap library. It can read from a device or a file and return a pointer to a received packet.

```
#include <PCAP.h>
```

### Public Member Functions

- [TPCAP](#) (void)
- void [openNIC](#) (char \*dev)
- void [openFile](#) (char \*name)
- void [openDevice](#) (char \*device, int live)
- `int8_t` [applyFilter](#) (char \*filter)
- `int8_t` [getPacketEvent](#) (void)
- void [close](#) (char \*dm, int dest)

## Public Attributes

- uint64\_t [TimeStamp](#)  
*Time of Packet-arrival in useconds sinze 1970.*
- uint32\_t [Length](#)  
*Number of available packet-bytes.*
- const uint8\_t \* [Packet](#)  
*Pointer the datastructure of the received Packet.*

## Private Attributes

- pcap\_t \* [Handle](#)  
*Handle to openened NIC or file.*
- char \* [DeviceName](#)  
*Name of the Device or File, like "eth0".*
- char \* [Filter](#)  
*Packet filter e.g. "not port 22" , default is "ip".*

### 5.16.1 Detailed Description

The "singleton" object of [TPCAP](#) is a wrapper around the libpcap library. It can read from a device or a file and return a pointer to a received packet.

<

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 TPCAP::TPCAP ( void )

The constructor.

### 5.16.3 Member Function Documentation

#### 5.16.3.1 int8\_t TPCAP::applyFilter ( char \* filter )

Applies a libpcap preprocessor filter.

#### Returns

0=ok

#### Parameters

<i>filter</i>	The pointer to a string with the filter equation e.g. "IP".
---------------	---

Here is the caller graph for this function:

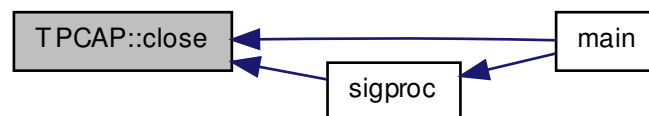


#### 5.16.3.2 void TPCAP::close ( char \* *dm*, int *dest* )

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.16.3.3 int8\_t TPCAP::getPacketEvent ( void )

Event check with immediate return, for use in a event loop. ;

**Returns**

0=no event, 1=packet received, -1=error, -2=EOF

Here is the caller graph for this function:

**5.16.3.4 void TPCAP::openDevice ( char \* device, int live )**

Opens a libcap-file or network device with recorded traffic.

If the name ends with a number an interface is assumed.

In other cases a file is assumed.

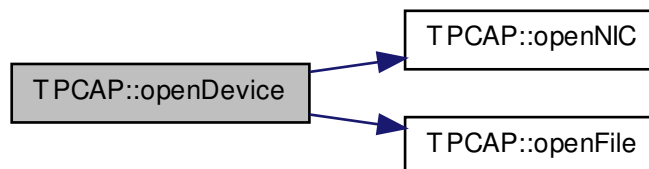
**Returns**

void

**Parameters**

<i>live</i>	0=from file, 1=from NIC
<i>device</i>	The pointer to a string with the name of the interface or file.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.16.3.5 void TPCAP::openFile ( char \* *name* )

Opens a libpcap file with recorded traffic e.g. "trace.pcap".

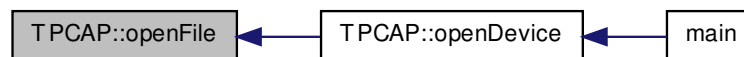
##### Returns

void

##### Parameters

<i>name</i>	The pointer to a string with the name of the file.
-------------	--

Here is the caller graph for this function:



#### 5.16.3.6 void TPCAP::openNIC ( char \* *dev* )

Opens a network device (interface) e.g. "eth0".

##### Returns

void

## Parameters

<i>dev</i>	The pointer to a string with the name of the device.
------------	--

Here is the caller graph for this function:



#### 5.16.4 Member Data Documentation

##### 5.16.4.1 `char* TPCAP::DeviceName` [private]

Name of the Device or File, like "eth0".

##### 5.16.4.2 `char* TPCAP::Filter` [private]

Packet filter e.g. "not port 22" , default is "ip".

##### 5.16.4.3 `pcap_t* TPCAP::Handle` [private]

Handle to openened NIC or file.

##### 5.16.4.4 `uint32_t TPCAP::Length`

Number of available packet-bytes.

##### 5.16.4.5 `const uint8_t* TPCAP::Packet`

Pointer the datastructure of the received Packet.

##### 5.16.4.6 `uint64_t TPCAP::TimeStamp`

Time of Packet-arrival in useconds sinze 1970.

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

- [src/PCAP.h](#)
- [src/PCAP.cpp](#)

## 5.17 TRPTDNSEvent Class Reference

```
#include <EventCollector.h>
```

### Public Member Functions

- [TRPTDNSEvent](#) (void)  
*The constructor. Clears all the data-members.*
- void [clear](#) (void)  
*Clears all data-members.*
- void [print](#) (char \*content)  
*Prints a cause in readable format to stdout.*

### Public Attributes

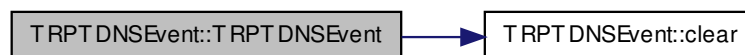
- int64\_t [TimeStamp](#)  
*TimeStamp: Time since 1970 in us.*
- char [Name](#) [64]  
*domain name in lowercase ending with '/0'(<64)*
- int32\_t [FlowIndex](#)  
*Index to flow that contains the event.*
- uint8\_t [CauseCount](#)  
*number of times the event caused new flows*

### 5.17.1 Constructor & Destructor Documentation

#### 5.17.1.1 TRPTDNSEvent::TRPTDNSEvent ( void )

The constructor. Clears all the data-members.

Here is the call graph for this function:





### 5.17.2 Member Function Documentation

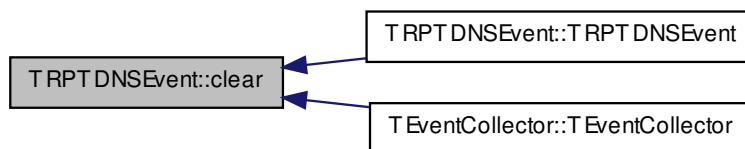
#### 5.17.2.1 void TRPTDNSEvent::clear ( void )

Clears all data-members.

##### Returns

void

Here is the caller graph for this function:



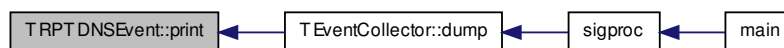
#### 5.17.2.2 void TRPTDNSEvent::print ( char \* content )

Prints a cause in readable format to stdout.

##### Returns

void

Here is the caller graph for this function:



### 5.17.3 Member Data Documentation

#### 5.17.3.1 uint8\_t TRPTDNSEvent::CauseCount

number of times the event caused new flows

### 5.17.3.2 `int32_t TRPTDNSEvent::FlowIndex`

Index to flow that contains the event.

### 5.17.3.3 `char TRPTDNSEvent::Name[64]`

domain name in lowercase ending with `'\0'` (<64)

### 5.17.3.4 `int64_t TRPTDNSEvent::TimeStamp`

TimeStamp: Time since 1970 in us.

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

- [src/EventCollector.h](#)
- [src/EventCollector.cpp](#)

## 5.18 TSettings Class Reference

Class for parsing and distributing the settings of the application.

### 5.18.1 Detailed Description

Class for parsing and distributing the settings of the application.

<

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

- [src/Settings.h](#)

## 5.19 TSettings Class Reference

```
#include <Settings.h>
```

### Public Member Functions

- [TSettings](#) (const char \*name)
- int [parseFile](#) (void)
- int [testWhiteList](#) (char \*id)
- int [testWhiteList](#) (uint32\_t ip)
- void [dump](#) (char \*content, int dest)

### Public Attributes

- int64\_t [DefinedValues](#) [SETTINGSSIZE]
- const char \* [DefinedSettings](#) [SETTINGSSIZE]
- uint32\_t [WhitePLow](#) [256]
- uint32\_t [WhitePHigh](#) [256]
- char [WhiteName](#) [256][256]

### Private Attributes

- char [Name](#) [100]
- int [WhiteIPTotal](#)
- int [WhiteNameTotal](#)

## 5.19.1 Constructor & Destructor Documentation

### 5.19.1.1 TSettings::TSettings ( const char \* *name* )

The constructor. Opens file

#### Parameters

<i>name</i>	filename with the settings
-------------	----------------------------

Here is the call graph for this function:



## 5.19.2 Member Function Documentation

### 5.19.2.1 void TSettings::dump ( char \* *content*, int *dest* )

Dumps the settings to stats file

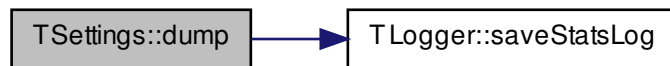
#### Returns

void

## Parameters

<i>*content</i>	pointer to content that must be printed. NULL is print to stdout
<i>dest</i>	destination of the log: 0= to stdout, 1=to logfile

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.19.2.2 int TSettings::parseFile ( void )

Parses the settings file.

## Returns

number of recognized lines or -1 if error

Here is the caller graph for this function:



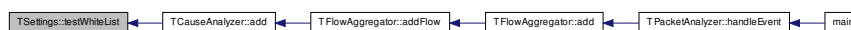
### 5.19.2.3 int TSettings::testWhiteList ( char \* *id* )

Tests if ID is in the whitelist

#### Returns

0=not in whitelist 1=in whitelist

Here is the caller graph for this function:



### 5.19.2.4 int TSettings::testWhiteList ( uint32\_t *ip* )

Tests if ip is in the whitelist

#### Returns

0=not in whitelist 1=in whitelist

## 5.19.3 Member Data Documentation

### 5.19.3.1 const char\* TSettings::DefinedSettings[SETTINGSSIZE]

### 5.19.3.2 int64\_t TSettings::DefinedValues[SETTINGSSIZE]

### 5.19.3.3 char TSettings::Name[100] [private]

Stores the Filename

### 5.19.3.4 uint32\_t TSettings::WhitelPHigh[256]

### 5.19.3.5 uint32\_t TSettings::WhitelPLow[256]

### 5.19.3.6 int TSettings::WhitelPTotal [private]

Total Number of Whitelist IP-ranges

### 5.19.3.7 char TSettings::WhiteName[256][256]

### 5.19.3.8 int TSettings::WhiteNameTotal [private]

Total Number of Whitelist Names

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

- [src/Settings.h](#)
- [src/Settings.cpp](#)

## 5.20 TTree Class Reference

Storage class for a tree of network flows with a causal relationship.

```
#include <Tree.h>
```

### Public Member Functions

- [TTree](#) (void)
- void [clear](#) (void)
- void [print](#) (char \*content)

### Public Attributes

- int64\_t [StartTime](#)  
*Timestamp of the root flow.*
- int64\_t [StopTime](#)  
*Timestamp of any last activity of a flow in the tree.*
- int32\_t [RootFlow](#)  
*ID of the root flow.*
- int8\_t [RootCause](#)  
*Cause of the root flow.*
- uint16\_t [NumberOfFlows](#)  
*Total number of flows (root + all offspring), 0 = tree empty.*
- uint16\_t [MaxDepth](#)  
*Longest line of descendants.*
- uint8\_t [DNSOther](#)  
*Default zero. Will be 1 if there if at least one other flow than a resolver flow.*
- char [ID](#) [256]  
*ID of the root Flow.*

### 5.20.1 Detailed Description

Storage class for a tree of network flows with a causal relationship.

<

## 5.20.2 Constructor & Destructor Documentation

### 5.20.2.1 TTree::TTree ( void )

The constructor. Resets all the data-members, including NextFlow

Here is the call graph for this function:



## 5.20.3 Member Function Documentation

### 5.20.3.1 void TTree::clear ( void )

Clears all the data-members.

Returns

void

Here is the caller graph for this function:



### 5.20.3.2 void TTree::print ( char \* content )

Prints a tree in readable format to stdout

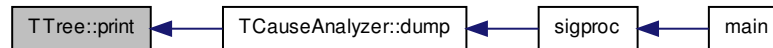
Returns

void

## Parameters

<i>*content</i>	pointer to content that must be printed. NULL is print to stdout
-----------------	--

Here is the caller graph for this function:



## 5.20.4 Member Data Documentation

### 5.20.4.1 uint8\_t TTree::DNSOther

Default zero. Will be 1 if there if at least one other flow than a resolver flow.

### 5.20.4.2 char TTree::ID[256]

ID of the root Flow.

### 5.20.4.3 uint16\_t TTree::MaxDepth

Longest line of descendants.

### 5.20.4.4 uint16\_t TTree::NumberOfFlows

Total number of flows (root + all offspring), 0 = tree empty.

### 5.20.4.5 int8\_t TTree::RootCause

Cause of the root flow.

### 5.20.4.6 int32\_t TTree::RootFlow

ID of the root flow.

### 5.20.4.7 int64\_t TTree::StartTime

Timestamp of the root flow.



## 5.20.4.8 int64\_t TTree::StopTime

Timestamp of any last activity of a flow in the tree.

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

- [src/Tree.h](#)
- [src/Tree.cpp](#)

## 5.21 TUDPUA Class Reference

One object of this class (class should be used as singleton) receives and filters user events, received from UDP with a special agent. Received results can be polled in a Event-loop.

```
#include <UDPUA.h>
```

### Public Member Functions

- [TUDPUA](#) (void)
- uint8\_t [open](#) (void)
- void [close](#) (void)
- uint8\_t [getEvent](#) (void)
- int [processEvent](#) (void)
- void [dump](#) (void)

### Public Attributes

- char [Message](#) [256]  
*Received raw message, only used in an opened socket after succesful getEvent.*
- char [Event](#) [256]  
*String, describing the event (LMB=Left Mouse Button, Enter, F5).*
- char [Process](#) [256]  
*String, describing the generating process that generates the user event.*
- int64\_t [TimeStamp](#)  
*TimeStamp: Time since 1970 in us.*
- uint8\_t [EventCode](#)  
*EventNumber: 0=none, 1=F5, 2=LMB, 3=Enter.*

### Private Attributes

- int [SocketDescriptor](#)  
*Socket descriptor for UDP-communication, listening on port UDP\_Port.*

### 5.21.1 Detailed Description

One object of this class (class should be used as singleton) receives and filters user events, received from UDP with a special agent. Received results can be polled in a Event-loop.

### 5.21.2 Constructor & Destructor Documentation

#### 5.21.2.1 TUDPUA::TUDPUA ( void )

The constructor that defines the UDP connection.

Clears the strings and EventCode;

### 5.21.3 Member Function Documentation

#### 5.21.3.1 void TUDPUA::close ( void )

Closes the UDP-socket.

##### Returns

void

#### 5.21.3.2 void TUDPUA::dump ( void )

Prints User Events with Timestamp, Event and Process.

##### Returns

void

#### 5.21.3.3 uint8\_t TUDPUA::getEvent ( void )

Reads UDP received events from an opened socket

##### Returns

0 if no event and something or 1 if something is received.

#### 5.21.3.4 uint8\_t TUDPUA::open ( void )

Binds the UDP-socket.

##### Returns

0 = ok 1+ = problem

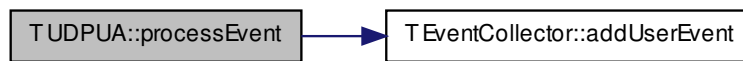
### 5.21.3.5 int TUDPUA::processEvent ( void )

Reads potential UDP received events from PCAP-packet. Remember that it will not use an open socket. Thi can result in ICMP-responses in the pcap-trace.

#### Returns

0=format ok but no event, 1=format ok and event, -1 = format not ok

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.21.4 Member Data Documentation

### 5.21.4.1 char TUDPUA::Event[256]

String, describing the event (LMB=Left Mouse Button, Enter, F5).

### 5.21.4.2 uint8\_t TUDPUA::EventCode

EventNumber: 0=none, 1=F5, 2=LMB, 3=Enter.

### 5.21.4.3 char TUDPUA::Message[256]

Received raw message, only used in an opened socket after succesful `getEvent`.

#### 5.21.4.4 char TUDPUA::Process[256]

String, describing the generating process that generates the user event.

#### 5.21.4.5 int TUDPUA::SocketDescriptor [private]

Socket descriptor for UDP-communication, listening on port UDP\_Port.

#### 5.21.4.6 int64\_t TUDPUA::TimeStamp

TimeStamp: Time since 1970 in us.

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

- src/UDPUA.h
- src/UDPUA.cpp

## 5.22 TURLEvent Class Reference

```
#include <EventCollector.h>
```

### Public Member Functions

- [TURLEvent](#) (void)  
*The constructor. Clears all the data-members.*
- void [clear](#) (void)  
*Clears all data-members.*
- void [print](#) (char \*content)  
*Prints a cause in readable format to stdout.*

### Public Attributes

- int64\_t [TimeStamp](#)  
*TimeStamp: Time since 1970 in us.*
- char [URL](#) [64]  
*domain name in lowercase ending with '/0'(<64)*
- char [SUBURL](#) [64]  
*2nd level domain name substring for softcompare*
- int32\_t [FlowIndex](#)  
*Index to flow that contains the event.*
- uint8\_t [CauseCount](#)  
*number of times the event caused new flows*

### 5.22.1 Constructor & Destructor Documentation

#### 5.22.1.1 TURLEvent::TURLEvent ( void )

The constructor. Clears all the data-members.

Here is the call graph for this function:



### 5.22.2 Member Function Documentation

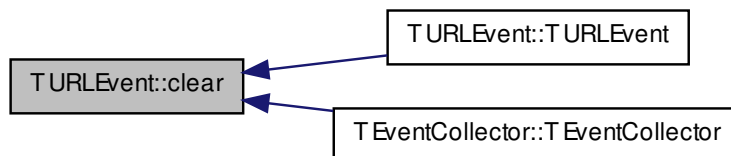
#### 5.22.2.1 void TURLEvent::clear ( void )

Clears all data-members.

Returns

void

Here is the caller graph for this function:



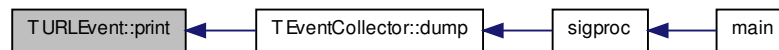
#### 5.22.2.2 void TURLEvent::print ( char \* content )

Prints a cause in readable format to stdout.

**Returns**

void

Here is the caller graph for this function:

**5.22.3 Member Data Documentation****5.22.3.1 uint8\_t TURLEvent::CauseCount**

number of times the event caused new flows

**5.22.3.2 int32\_t TURLEvent::FlowIndex**

Index to flow that contains the event.

**5.22.3.3 char TURLEvent::SUBURL[64]**

2nd level domain name substring for softcompare

**5.22.3.4 int64\_t TURLEvent::TimeStamp**

TimeStamp: Time since 1970 in us.

**5.22.3.5 char TURLEvent::URL[64]**

domain name in lowercase ending with '/0'(<64)

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

- [src/EventCollector.h](#)
- [src/EventCollector.cpp](#)

**5.23 TUserEvent Class Reference**

```
#include <EventCollector.h>
```

## Public Member Functions

- [TUserEvent](#) (void)  
*The constructor. Clears all the data-members.*
- void [clear](#) (void)  
*Clears all data-members.*
- void [print](#) (char \*content)  
*Prints a cause in readable format to stdout.*

## Public Attributes

- int64\_t [TimeStamp](#)  
*TimeStamp: Time since 1970 in us.*
- uint8\_t [EventCode](#)  
*EventNumber: 0=none, 1=F5, 2=LMB, 3=Enter.*
- char [Process](#) [64]  
*String, describing the generating process that generates the user event in lowercase ending with '\0'(<64)*
- uint8\_t [CauseCount](#)  
*number of times the event caused new flows*

### 5.23.1 Constructor & Destructor Documentation

#### 5.23.1.1 TUserEvent::TUserEvent ( void )

The constructor. Clears all the data-members.

Here is the call graph for this function:



### 5.23.2 Member Function Documentation

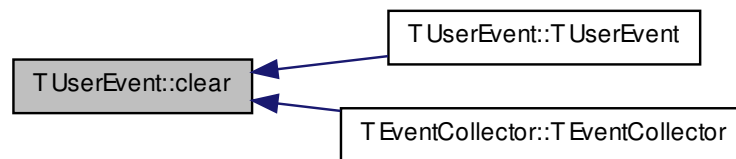
#### 5.23.2.1 void TUserEvent::clear ( void )

Clears all data-members.

**Returns**

void

Here is the caller graph for this function:

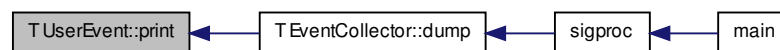
**5.23.2.2 void TUserEvent::print ( char \* *content* )**

Prints a cause in readable format to stdout.

**Returns**

void

Here is the caller graph for this function:

**5.23.3 Member Data Documentation****5.23.3.1 uint8\_t TUserEvent::CauseCount**

number of times the event caused new flows

**5.23.3.2 uint8\_t TUserEvent::EventCode**

EventNumber: 0=none, 1=F5, 2=LMB, 3=Enter.



## 5.23.3.3 char TUserEvent::Process[64]

String, describing the generating process that generates the user event in lowercase ending with '\0'(<64)

## 5.23.3.4 int64\_t TUserEvent::TimeStamp

TimeStamp: Time since 1970 in us.

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

- src/[EventCollector.h](#)
- src/[EventCollector.cpp](#)

## 5.24 TUTreeEvent Class Reference

```
#include <EventCollector.h>
```

## Public Member Functions

- [TUTreeEvent](#) (void)  
*The constructor. Clears all the data-members.*
- void [clear](#) (void)  
*Clears all data-members.*
- void [print](#) (char \*content)  
*Prints a cause in readable format to stdout.*

## Public Attributes

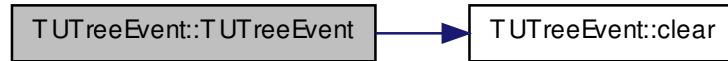
- int64\_t [TimeStamp](#)  
*TimeStamp: Time since 1970 in us.*
- int32\_t [TreeIndex](#)  
*Index to tree that contains the event.*
- uint8\_t [CauseCount](#)  
*number of times the event caused new flows*
- int64\_t [DeltaT](#)  
*Time contributed to openwindow.*

## 5.24.1 Constructor &amp; Destructor Documentation

## 5.24.1.1 TUTreeEvent::TUTreeEvent ( void )

The constructor. Clears all the data-members.

Here is the call graph for this function:



## 5.24.2 Member Function Documentation

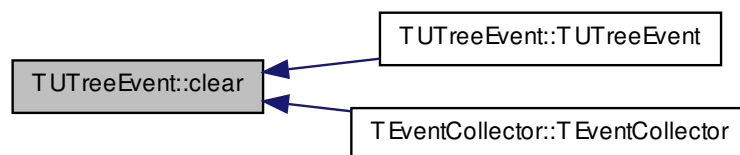
### 5.24.2.1 void TUTreeEvent::clear ( void )

Clears all data-members.

#### Returns

void

Here is the caller graph for this function:

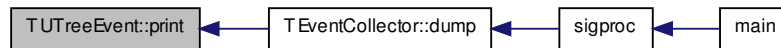


### 5.24.2.2 void TUTreeEvent::print ( char \* content )

Prints a cause in readable format to stdout.

### Returns

Here is the caller graph for this function:



## 5.24.3 Member Data Documentation

### 5.24.3.1 `uint8_t TUTreeEvent::CauseCount`

number of times the event caused new flows

### 5.24.3.2 `int64_t TUTreeEvent::DeltaT`

Time contributed to openwindow.

### 5.24.3.3 `int64_t TUTreeEvent::TimeStamp`

TimeStamp: Time since 1970 in us.

### 5.24.3.4 `int32_t TUTreeEvent::TreeIndex`

Index to tree that contains the event.

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

- [src/EventCollector.h](#)
- [src/EventCollector.cpp](#)



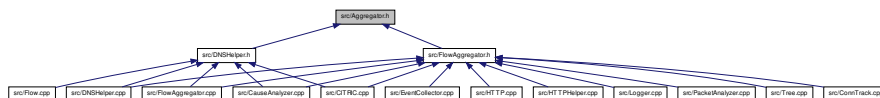
## Chapter 6

# File Documentation

### 6.1 src/Aggregator.h File Reference

This file contains the base class [TAggregator](#).

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



### Classes

- class [TAggregator](#)

*Baseclass for aggregators. An aggregator is an object that aggregates multiple data in a combined datastructure or element of a small size. Examples of aggregating elements are Flows, Tree, and DNS-records.*

#### 6.1.1 Detailed Description

This file contains the base class [TAggregator](#).

### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

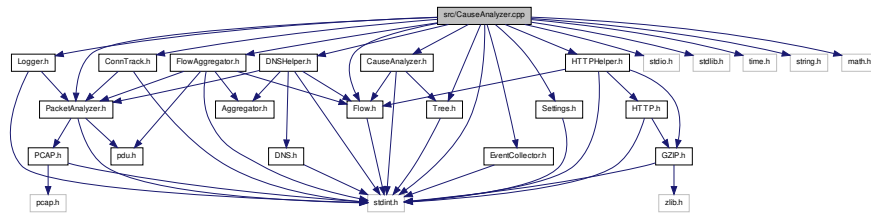
**Date**

Saturday, October 20, 2012

## 6.2 src/CauseAnalyzer.cpp File Reference

This file contains the operators of the [TCauseAnalyzer](#) class.

```
#include <stdint.h> #include <stdio.h> #include <stdlib.-
h> #include <time.h> #include <string.h> #include <math.-
h> #include "FlowAggregator.h" #include "Flow.h" #include
"DNSHelper.h" #include "Tree.h" #include "CauseAnalyzer.-
h" #include "HTTPHelper.h" #include "PacketAnalyzer.h" ×
#include "EventCollector.h" #include "Logger.h" #include
"Settings.h" #include "ConnTrack.h" Include dependency graph for
CauseAnalyzer.cpp:
```

**Variables**

- [TFlowAggregator](#) \* [FlowAggregator](#)
- [TDNSHelper](#) \* [DNSHelper](#)
- [TPCAP](#) \* [PCAP](#)
- [TPacketAnalyzer](#) \* [PacketAnalyzer](#)
- [TFlow](#) [Flow](#) [[FLOWBUFFERSIZE](#)]
- [TDNS](#) [DNS](#) [[DNSBUFFERSIZE](#)]
- [THTTP](#) [HTTP](#) [[HTTP\\_BUFFER\\_SIZE](#)]
- [THTTPHelper](#) \* [HTTPHelper](#)
- [TEventCollector](#) \* [EventCollector](#)
- [TLogger](#) \* [Logger](#)

- [TSettings](#) \* [Settings](#)
- [TConnTrack](#) \* [ConnTrack](#)
- int [ENDPROG](#)
- [TTree](#) [Tree](#) [[TREE\\_BUFFER\\_SIZE](#)]

### 6.2.1 Detailed Description

This file contains the operators of the [TCauseAnalyzer](#) class.

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Tuesday, March, 12, 2013

### 6.2.2 Variable Documentation

- 6.2.2.1 [TConnTrack](#)\* [ConnTrack](#)
- 6.2.2.2 [TDNS](#) [DNS](#)[[DNSBUFFERSIZE](#)]
- 6.2.2.3 [TDNSHelper](#)\* [DNSHelper](#)
- 6.2.2.4 int [ENDPROG](#)
- 6.2.2.5 [TEventCollector](#)\* [EventCollector](#)
- 6.2.2.6 [TFlow](#) [Flow](#)[[FLOWBUFFERSIZE](#)]
- 6.2.2.7 [TFlowAggregator](#)\* [FlowAggregator](#)
- 6.2.2.8 [THTTP](#) [HTTP](#)[[HTTP\\_BUFFER\\_SIZE](#)]
- 6.2.2.9 [THTTPHelper](#)\* [HTTPHelper](#)
- 6.2.2.10 [TLogger](#)\* [Logger](#)
- 6.2.2.11 [TPacketAnalyzer](#)\* [PacketAnalyzer](#)
- 6.2.2.12 [TPCAP](#)\* [PCAP](#)

### 6.2.2.13 TSettings\* Settings

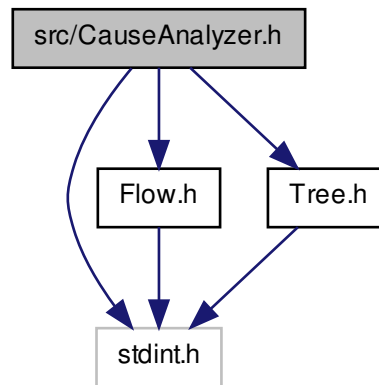
### 6.2.2.14 TTree Tree[TREE\_BUFFER\_SIZE]

## 6.3 src/CauseAnalyzer.h File Reference

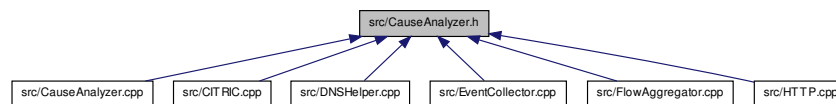
This file contains the prototypes of the [TCauseAnalyzer](#) class.

```
#include <stdint.h> #include "Flow.h" #include "Tree.h" ×
```

Include dependency graph for CauseAnalyzer.h:



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



## Classes

- class [TCauseAnalyzer](#)

*Class that manages causal trees and sorts new flows in it.*



## Defines

- #define [TREE\\_BUFFER\\_SIZE](#) 65536

### 6.3.1 Detailed Description

This file contains the prototypes of the [TCauseAnalyzer](#) class.

## Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

## Author

Pieter Burghouwt

## Version

Revision 2.0

## Date

Tuesday, March, 12, 2013

### 6.3.2 Define Documentation

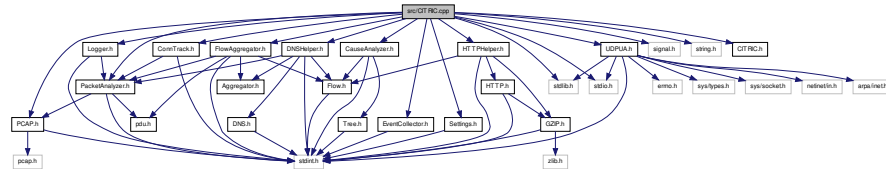
#### 6.3.2.1 #define [TREE\\_BUFFER\\_SIZE](#) 65536

## 6.4 src/CITRIC.cpp File Reference

This file contains the main file of CITRIC. CITRIC is an experimental causal detector that analyzes PCAP-data from a device. It aggregates packets in bidirectional flows and it organizes the flows in causal trees. A causal tree is a group of flows with a causal relationship. Causal relationship is determined by time-intervals and in some cases content. This is an experimental version for research.

```
#include <stdlib.h> #include <stdio.h> #include <signal.-  
h> #include <string.h> #include "PCAP.h" #include "Packet-  
Analyzer.h" #include "FlowAggregator.h" #include "DN-  
SHelper.h" #include "CauseAnalyzer.h" #include "HTTP-  
Helper.h" #include "EventCollector.h" #include "UDPUA.-  
h" #include "CITRIC.h" #include "Logger.h" #include "-
```

```
Settings.h" #include "ConnTrack.h" Include dependency graph for -
CITRIC.cpp:
```



## Functions

- int **main** (int argc, char \*argv[])
- void **sigproc** (int signum)

## Variables

- [TPCAP](#) \* [PCAP](#)
- [TPacketAnalyzer](#) \* [PacketAnalyzer](#)
- [TFlowAggregator](#) \* [FlowAggregator](#)
- [TDNSHelper](#) \* [DNSHelper](#)
- [TCauseAnalyzer](#) \* [CauseAnalyzer](#)
- [THTTPHelper](#) \* [HTTPHelper](#)
- [TUDPUA](#) \* [UDPUA](#)
- [TEventCollector](#) \* [EventCollector](#)
- [TLogger](#) \* [Logger](#)
- [TSettings](#) \* [Settings](#)
- [TConnTrack](#) \* [ConnTrack](#)
- char [DumpMessage](#) [100000000]
- int [ENDPROG](#)

### 6.4.1 Detailed Description

This file contains the main file of CITRIC. CITRIC is an experimental causal detector that analyzes PCAP-data from a device. It aggregates packets in bidirectional flows and it organizes the flows in causal trees. A causal tree is a group of flows with a causal relationship. Causal relationship is determined by time-intervals and in some cases content. This is an experimental version for research.

## Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

Pieter Burghouwt

Revision 2.0

Friday, November 16, 2012

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

A handler for OS-signals, to catch the CTRL+C

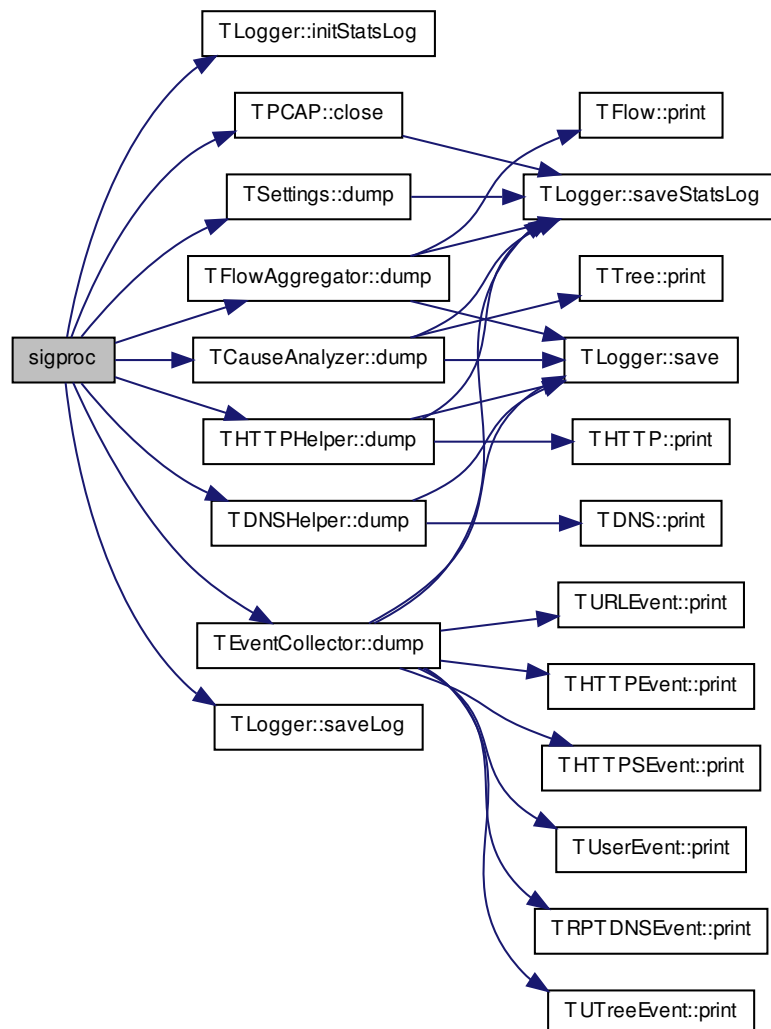
## Returns

void

## Parameters

<i>signum</i>	=2 (SIGINT) that identifies to CTRL+C
---------------	---------------------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.4.3 Variable Documentation

6.4.3.1 `TCauseAnalyzer*` `CauseAnalyzer`

6.4.3.2 `TConnTrack*` `ConnTrack`

6.4.3.3 `TDNSHelper*` `DNSHelper`

6.4.3.4 `char DumpMessage[100000000]`

6.4.3.5 `int ENDPROG`

6.4.3.6 `TEventCollector*` `EventCollector`

6.4.3.7 `TFlowAggregator*` `FlowAggregator`

6.4.3.8 `THTTPHelper*` `HTTPHelper`

6.4.3.9 `TLogger*` `Logger`

6.4.3.10 `TPacketAnalyzer*` `PacketAnalyzer`

6.4.3.11 `TPCAP*` `PCAP`

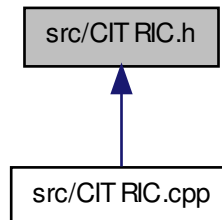
6.4.3.12 `TSettings*` `Settings`

6.4.3.13 `TUDPUA*` `UDPUA`

## 6.5 src/CITRIC.h File Reference

This file contains the prototypes of the CITRIC class.

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



## Functions

- void `sigproc` (int signum)

### 6.5.1 Detailed Description

This file contains the prototypes of the CITRIC class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Friday, November 16, 2012

### 6.5.2 Function Documentation

#### 6.5.2.1 void `sigproc` ( int *signum* )

A handler for OS-signals, to catch the CTRL+C

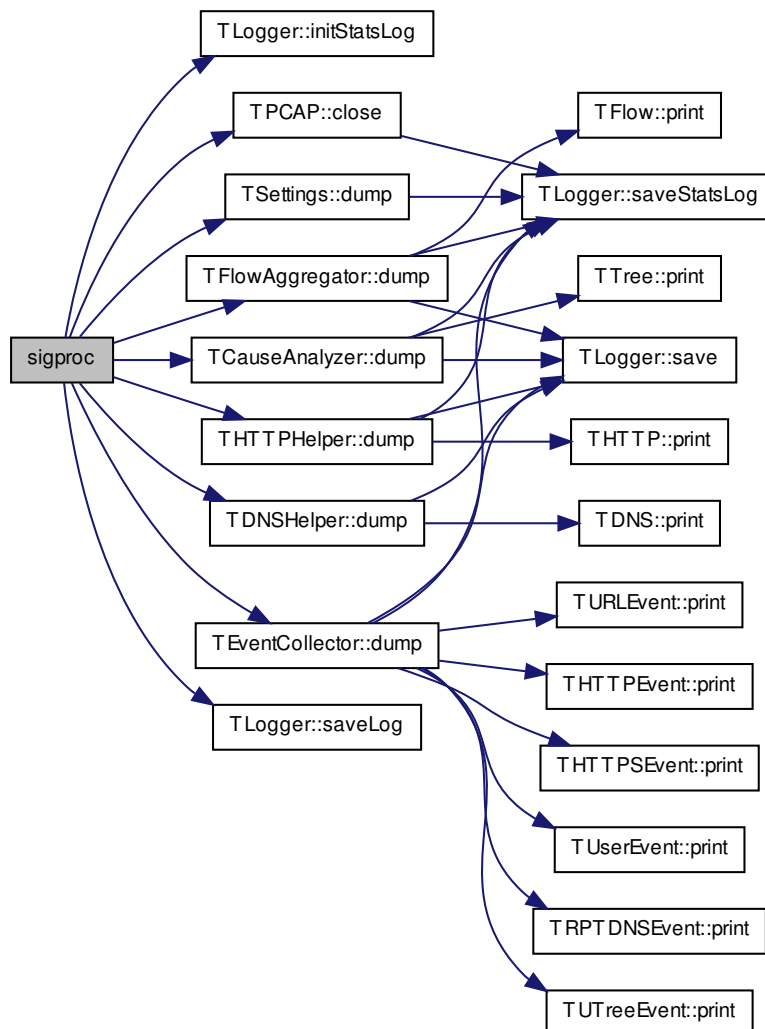
## Returns

void

## Parameters

<i>signum</i>	=2 (SIGINT) that identifies to CTRL+C
---------------	---------------------------------------

Here is the call graph for this function:



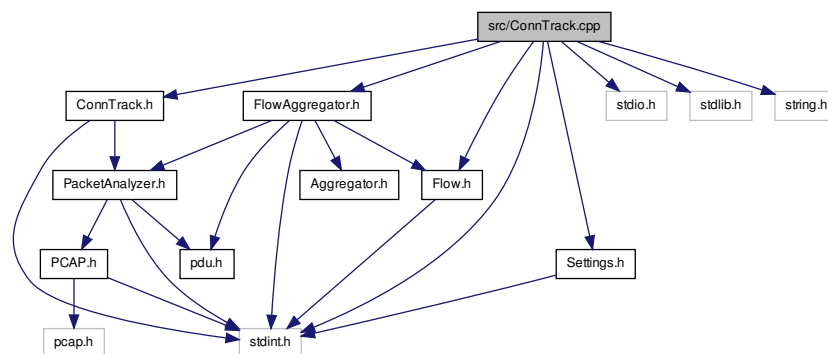
Here is the caller graph for this function:



## 6.6 src/ConnTrack.cpp File Reference

This file contains the operators of the [TConnTrack](#) class.

```
#include <stdint.h> #include <stdio.h> #include <stdlib.-
h> #include <string.h> #include "Flow.h" #include "Flow-
Aggregator.h" #include "ConnTrack.h" #include "Settings.-
h" Include dependency graph for ConnTrack.cpp:
```



### Variables

- [TSettings](#) \* [Settings](#)
- [TFlow](#) [Flow](#) [[FLOWBUFFERSIZE](#)]

### 6.6.1 Detailed Description

This file contains the operators of the [TConnTrack](#) class.



**Copyright**

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

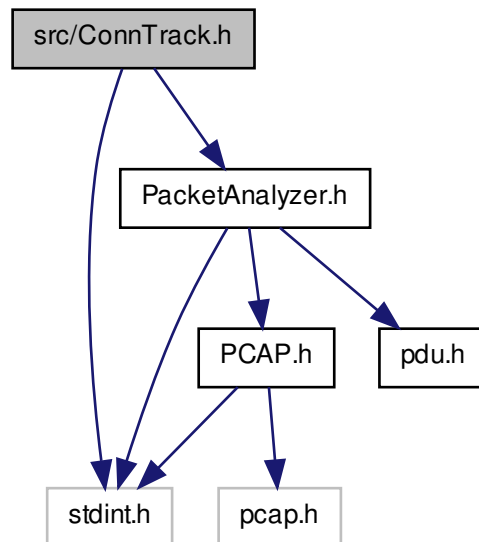
**Date**

Wednesday, January 2, 2013

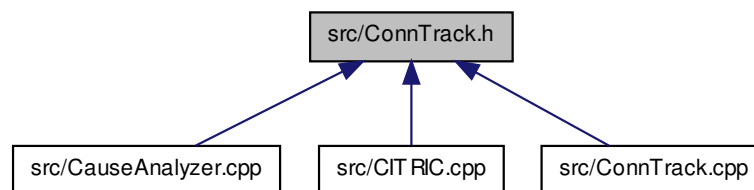
**6.6.2 Variable Documentation****6.6.2.1 TFlow Flow[FLOWBUFFERSIZE]****6.6.2.2 TSettings\* Settings****6.7 src/ConnTrack.h File Reference**

This file contains the prototypes of the [TConnTrack](#) class.

```
#include <stdint.h> #include "PacketAnalyzer.h" Include dependency graph for ConnTrack.h:
```



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



## Classes

- class [TConnTrack](#)

*Class that can kill an established connection.*

### 6.7.1 Detailed Description

This file contains the prototypes of the [TConnTrack](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

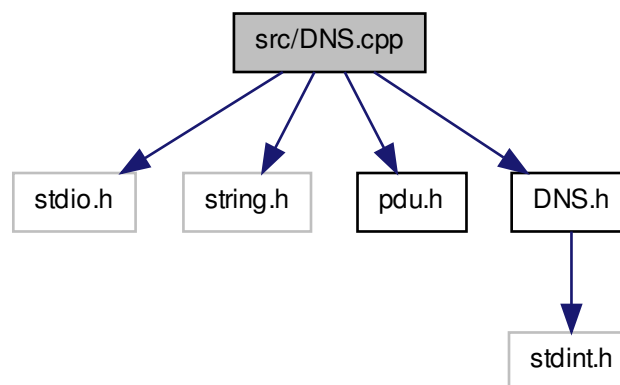
#### Date

Wednesday, January 1, 2013

## 6.8 src/DNS.cpp File Reference

This file contains the operators of the [TDNS](#) class.

```
#include <stdio.h> #include <string.h> #include "pdu.h" ×  
#include "DNS.h" Include dependency graph for DNS.cpp:
```



### 6.8.1 Detailed Description

This file contains the operators of the [TDNS](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

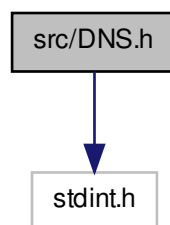
#### Date

Thursday, February 28, 2013

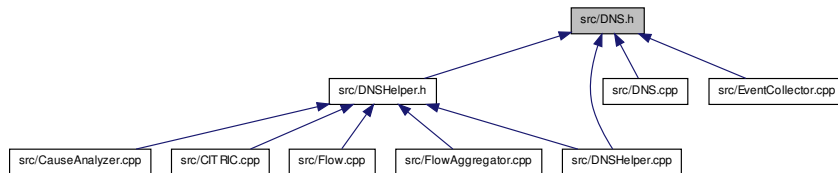
## 6.9 src/DNS.h File Reference

This file contains the prototypes of the [TDNS](#) class.

`#include <stdint.h>` Include dependency graph for DNS.h:



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



## Classes

- class [TDNS](#)

*Storage class for DNS-data. Objects of this class are managed by `DNSHelper`.*

## Defines

- `#define` [MAXDOMAINNAMELENGTH](#) 64

### 6.9.1 Detailed Description

This file contains the prototypes of the [TDNS](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Thursday, February 28, 2013

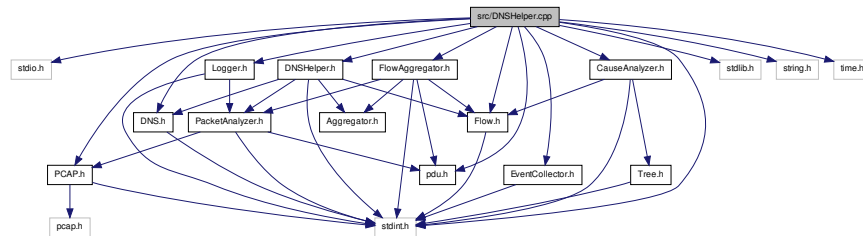
## 6.9.2 Define Documentation

### 6.9.2.1 #define MAXDOMAINNAMELENGTH 64

## 6.10 src/DNSHelper.cpp File Reference

This file contains the operators of the [TDNSHelper](#) class.

```
#include <stdio.h> #include <stdint.h> #include <stdlib.-
h> #include <string.h> #include <time.h> #include "pdu.-
h" #include "PCAP.h" #include "DNSHelper.h" #include "DNS.-
h" #include "FlowAggregator.h" #include "EventCollector.-
h" #include "CauseAnalyzer.h" #include "Flow.h" #include "-
Logger.h" Include dependency graph for DNSHelper.cpp:
```



## Variables

- [TPCAP](#) \* [PCAP](#)
- [TPacketAnalyzer](#) \* [PacketAnalyzer](#)
- [TFlowAggregator](#) \* [FlowAggregator](#)
- [TEventCollector](#) \* [EventCollector](#)
- [TCauseAnalyzer](#) \* [CauseAnalyzer](#)
- [TLogger](#) \* [Logger](#)
- [TFlow](#) [Flow](#) [[FLOWBUFFERSIZE](#)]
- [TDNS](#) [DNS](#) [[DNSBUFFERSIZE](#)]

### 6.10.1 Detailed Description

This file contains the operators of the [TDNSHelper](#) class.

## Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

**Date**

Thursday, February 28, 2013

**6.10.2 Variable Documentation**

**6.10.2.1 TCauseAnalyzer\* CauseAnalyzer**

**6.10.2.2 TDNS DNS[DNSBUFFERSIZE]**

**6.10.2.3 TEventCollector\* EventCollector**

**6.10.2.4 TFlow Flow[FLOWBUFFERSIZE]**

**6.10.2.5 TFlowAggregator\* FlowAggregator**

**6.10.2.6 TLogger\* Logger**

**6.10.2.7 TPacketAnalyzer\* PacketAnalyzer**

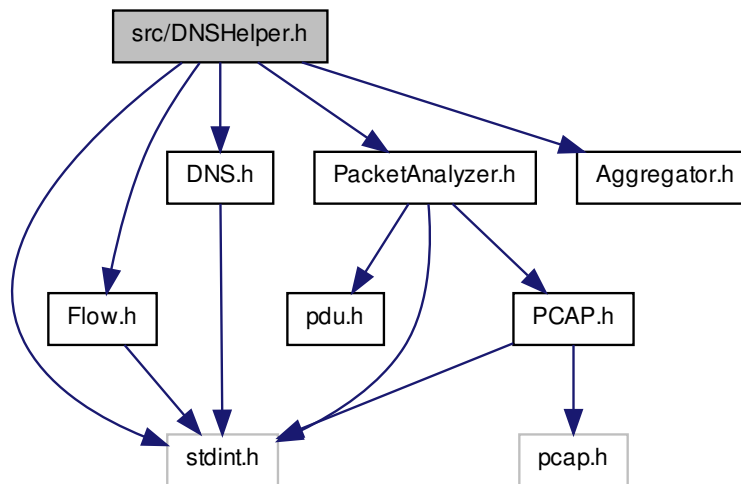
**6.10.2.8 TPCAP\* PCAP**

**6.11 src/DNSHelper.h File Reference**

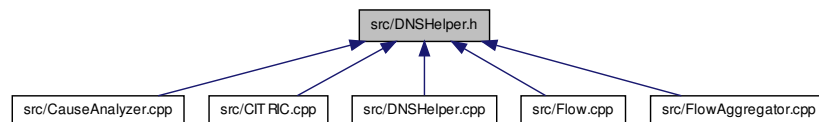
This file contains the prototypes of the [TDNSHelper](#) class.

```
#include <stdint.h> #include "PacketAnalyzer.h" #include  
"Flow.h" #include "DNS.h" #include "Aggregator.h" Include de-
```

pendency graph for DNSHelper.h:



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



## Classes

- class [TDNSHelper](#)

*Class specialized in aggregating DNS events. It only processes replies from egress UDP DNS-flows. DNS objects are stored as a chained hashtable of unidirectional linked lists. The hashtable is based on a modulo 256 sum of the IP-address. This results in a fast lookup by IP-address. Newer records with the same name and IP-address are replaced. TODO: cleanup-routine that checks one record at a time on expiry.*

## Defines

- `#define` [DNSBUFFERSIZE](#) 65536



*total buffer size of DNS-objects*

- `#define DNSHASHSIZE 1024`

*must be a power of 2. Max is  $2^{16}$*

### 6.11.1 Detailed Description

This file contains the prototypes of the [TDNSHelper](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Thursday, February 28, 2013

### 6.11.2 Define Documentation

#### 6.11.2.1 `#define DNSBUFFERSIZE 65536`

*total buffer size of DNS-objects*

#### 6.11.2.2 `#define DNSHASHSIZE 1024`

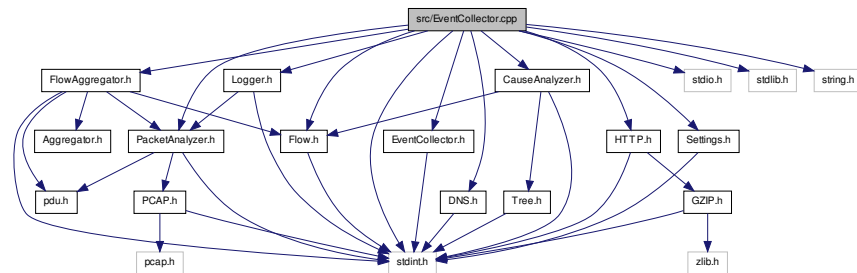
*must be a power of 2. Max is  $2^{16}$*

## 6.12 src/EventCollector.cpp File Reference

This file contains the operators of the EventCollector class.

```
#include <stdint.h> #include <stdio.h> #include <stdlib.h>
#include <string.h> #include "FlowAggregator.h" #include
```

```
"Flow.h" #include "HTTP.h" #include "EventCollector.h" ×
#include "DNS.h" #include "PacketAnalyzer.h" #include "-
Logger.h" #include "CauseAnalyzer.h" #include "Settings.-
h" Include dependency graph for EventCollector.cpp:
```



## Variables

- [TPacketAnalyzer](#) \* [PacketAnalyzer](#)
- [TFlowAggregator](#) \* [FlowAggregator](#)
- [TLogger](#) \* [Logger](#)
- [TFlow](#) [Flow](#) []
- [THTTP](#) [HTTP](#) []
- [TDNS](#) [DNS](#) []
- [TSettings](#) \* [Settings](#)

## 6.12.1 Detailed Description

This file contains the operators of the EventCollector class.

### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

### Author

Pieter Burghouwt

### Version

Revision 2.0

### Date

Monda, March 11, 2013

### 6.12.2 Variable Documentation

6.12.2.1 TDNS DNS[]

6.12.2.2 TFlow Flow[]

6.12.2.3 TFlowAggregator\* FlowAggregator

6.12.2.4 THTTP HTTP[]

6.12.2.5 TLogger\* Logger

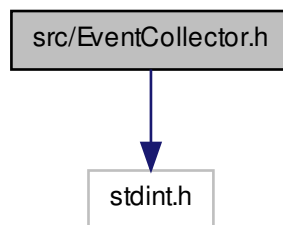
6.12.2.6 TPacketAnalyzer\* PacketAnalyzer

6.12.2.7 TSettings\* Settings

## 6.13 src/EventCollector.h File Reference

This file contains the prototypes of the [TEventCollector](#) class.

`#include <stdint.h>` Include dependency graph for EventCollector.h:



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



## Classes

- class [TUserEvent](#)
- class [THTTPEvent](#)
- class [THTTPSEvent](#)
- class [TURLEvent](#)
- class [TRPTDNSEvent](#)
- class [TUTreeEvent](#)
- class [TEventCollector](#)

*Class specialized in aggregating events that can cause new traffic. Every potential event is buffered here. Hashing is used for fast lookup. FIFO's are used to prevent overflow.*

## Defines

- #define [EVENTBUFFERSIZE](#) 10000
- #define [RPTDNSEVENTBUFFERSIZE](#) 100
- #define [URLEVENTBUFFERSIZE](#) 1000
- #define [URLEVENTHASHSIZE](#) 256
- #define [CAUSE\\_HTTPDATARECEIVED](#) 1
- #define [CAUSE\\_HTTPDATAPUSH](#) 2
- #define [CAUSE\\_HTTPSIZEDECREASE](#) 3
- #define [CAUSE\\_HTTPWITHDRAWN](#) 4

### 6.13.1 Detailed Description

This file contains the prototypes of the [TEventCollector](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Monday, March 11, 2013

### 6.13.2 Define Documentation

6.13.2.1 `#define CAUSE_HTTPDATAPUSH 2`

6.13.2.2 `#define CAUSE_HTTPDATARECEIVED 1`

6.13.2.3 `#define CAUSE_HTTPSIZEDECREASE 3`

6.13.2.4 `#define CAUSE_HTTPWITHDRAWN 4`

6.13.2.5 `#define EVENTBUFFERSIZE 10000`

6.13.2.6 `#define RPTDNSEVENTBUFFERSIZE 100`

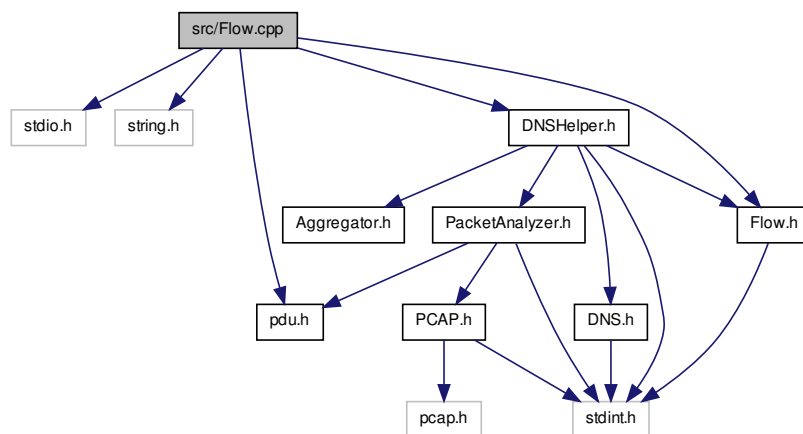
6.13.2.7 `#define URLEVENTBUFFERSIZE 1000`

6.13.2.8 `#define URLEVENTHASHSIZE 256`

## 6.14 src/Flow.cpp File Reference

This file contains the operators of the [TFlow](#) class.

```
#include <stdio.h> #include <string.h> #include "pdu.h" ×  
#include "Flow.h" #include "DNSHelper.h" Include dependency graph  
for Flow.cpp:
```



## Variables

- [TDNS DNS \[DNSBUFFERSIZE\]](#)

### 6.14.1 Detailed Description

This file contains the operators of the [TFlow](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 0.9

#### Date

Tuesday, March 12, 2013

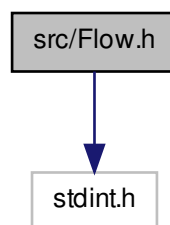
### 6.14.2 Variable Documentation

#### 6.14.2.1 TDNS DNS[DNSBUFFERSIZE]

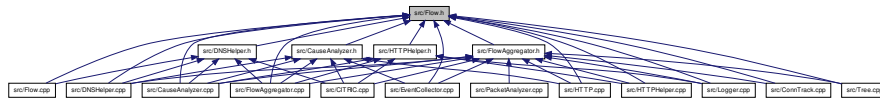
## 6.15 src/Flow.h File Reference

This file contains the prototypes of the [TFlow](#) class.

`#include <stdint.h>` Include dependency graph for Flow.h:



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



## Classes

- class [TFlow](#)  
*Storage class for a bidirectional flow.*

## Defines

- #define [FLOWHASHSIZE](#) 1024
- #define [UNDEFINED\\_DIR](#) 0
- #define [INGRESS](#) 1
- #define [EGRESS](#) 2
- #define [RESOLVER\\_UNDEFINED](#) 0
- #define [RESOLVER\\_PROTO\\_STATIC](#) 1
- #define [RESOLVER\\_PROTO\\_DNS](#) 2
- #define [RESOLVER\\_PROTO\\_OTHER](#) 3
- #define [RESOLVER\\_NONAME](#) 128
- #define [RESOLVER\\_STATICNAME](#) 129
- #define [RESOLVER\\_DNSNAME](#) 130
- #define [RESOLVER\\_OTHERNAME](#) 131
- #define [CAUSE\\_UNKNOWN](#) 0
- #define [CAUSE\\_SERVER](#) 1
- #define [CAUSE\\_DNS](#) 2
- #define [CAUSE\\_HTTP\\_URL](#) 3
- #define [CAUSE\\_HTTP\\_SOFTURL](#) 4
- #define [CAUSE\\_HTTP\\_REFERER](#) 5
- #define [CAUSE\\_HTTP\\_GEN](#) 6
- #define [CAUSE\\_HTTPS\\_GEN](#) 7
- #define [CAUSE\\_USER](#) 8
- #define [CAUSE\\_PROTODNS](#) 9
- #define [CAUSE\\_WHITELIST](#) 10
- #define [CAUSE\\_ALREADYOPEN](#) 11
- #define [CAUSE\\_DNS\\_REPEAT](#) 12
- #define [CAUSE\\_UTREE](#) 13
- #define [CAUSEQ\\_NOID\\_NOTIME](#) 0
- #define [CAUSEQ\\_NOID\\_SOFTTIME](#) 1
- #define [CAUSEQ\\_NOID\\_TIME](#) 2
- #define [CAUSEQ\\_TIME](#) 3

- #define CAUSEQ\_SOFTID\_NOTIME 4
- #define CAUSEQ\_SOFTID\_SOFTTIME 5
- #define CAUSEQ\_SOFTID\_TIME 6
- #define CAUSEQ\_ID\_NOTIME 7
- #define CAUSEQ\_ID\_SOFTTIME 8
- #define CAUSEQ\_ID\_TIME 9

### 6.15.1 Detailed Description

This file contains the prototypes of the [TFlow](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Tuesday, March 12, 2013

### 6.15.2 Define Documentation

6.15.2.1 #define CAUSE\_ALREADYOPEN 11

6.15.2.2 #define CAUSE\_DNS 2

6.15.2.3 #define CAUSE\_DNS\_REPEAT 12

6.15.2.4 #define CAUSE\_HTTP\_GEN 6

6.15.2.5 #define CAUSE\_HTTP\_REFERER 5

6.15.2.6 #define CAUSE\_HTTP\_SOFTURL 4

6.15.2.7 #define CAUSE\_HTTP\_URL 3

6.15.2.8 #define CAUSE\_HTTPS\_GEN 7



6.15.2.9    `#define CAUSE_PROTODNS 9`

6.15.2.10   `#define CAUSE_SERVER 1`

6.15.2.11   `#define CAUSE_UNKNOWN 0`

6.15.2.12   `#define CAUSE_USER 8`

6.15.2.13   `#define CAUSE_UTREE 13`

6.15.2.14   `#define CAUSE_WHITELIST 10`

6.15.2.15   `#define CAUSEQ_ID_NOTIME 7`

6.15.2.16   `#define CAUSEQ_ID_SOFTTIME 8`

6.15.2.17   `#define CAUSEQ_ID_TIME 9`

6.15.2.18   `#define CAUSEQ_NOID_NOTIME 0`

6.15.2.19   `#define CAUSEQ_NOID_SOFTTIME 1`

6.15.2.20   `#define CAUSEQ_NOID_TIME 2`

6.15.2.21   `#define CAUSEQ_SOFTID_NOTIME 4`

6.15.2.22   `#define CAUSEQ_SOFTID_SOFTTIME 5`

6.15.2.23   `#define CAUSEQ_SOFTID_TIME 6`

6.15.2.24   `#define CAUSEQ_TIME 3`

6.15.2.25   `#define EGRESS 2`

6.15.2.26   `#define FLOWHASHSIZE 1024`

6.15.2.27   `#define INGRESS 1`

6.15.2.28   `#define RESOLVER_DNSNAME 130`

6.15.2.29   `#define RESOLVER_NONAME 128`

6.15.2.30   `#define RESOLVER_OTHERNAME 131`

6.15.2.31   `#define RESOLVER_PROTO_DNS 2`

6.15.2.32   `#define RESOLVER_PROTO_OTHER 3`

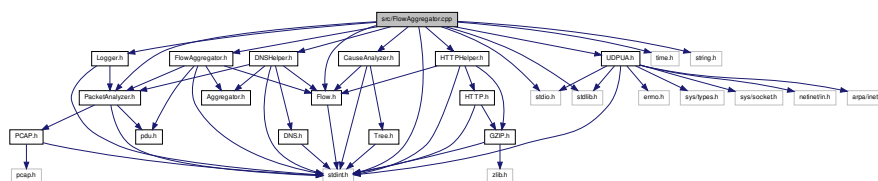
#### 6.15.2.34 #define RESOLVER\_STATICNAME 129

### 6.15.2.35 #define RESOLVER\_UNDEFINED 0

### 6.15.2.36 #define UNDEFINED DIR 0

This file contains the operators of the `TFlowAggregator` class.

```
#include <stdint.h> #include <stdio.h> #include <stdlib.h>
#include <time.h> #include <string.h> #include "Packet-
Analyzer.h" #include "FlowAggregator.h" #include "Flow.h"
#include "DNSHelper.h" #include "CauseAnalyzer.h" #include
"HTTPHelper.h" #include "UDPUA.h" #include "Logger.h" Include
dependency graph for FlowAggregator.cpp:
```



- `TPCAP * PCAP`
- `TPacketAnalyzer * PacketAnalyzer`
- `TDNSHelper * DNSHelper`
- `TCauseAnalyzer * CauseAnalyzer`
- `THTTPHelper * HTTPHelper`
- `TUDPUA * UDPUA`
- `TLogger * Logger`
- `int32_t FlowTable [FLOWHASHSIZE]`
- `TFlow Flow [FLOWBUFFERSIZE]`

This file contains the operators of the `TFlowAggregator` class.

**Copyright**

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

**Date**

Tuesday, March 12, 2013

**6.16.2 Variable Documentation**

6.16.2.1 **TCauseAnalyzer\*** CauseAnalyzer

6.16.2.2 **TDNSHelper\*** DNSHelper

6.16.2.3 **TFlow** Flow[FLOWBUFFERSIZE]

6.16.2.4 **int32\_t** FlowTable[FLOWHASHSIZE]

6.16.2.5 **THTTPHelper\*** HTTPHelper

6.16.2.6 **TLogger\*** Logger

6.16.2.7 **TPacketAnalyzer\*** PacketAnalyzer

6.16.2.8 **TPCAP\*** PCAP

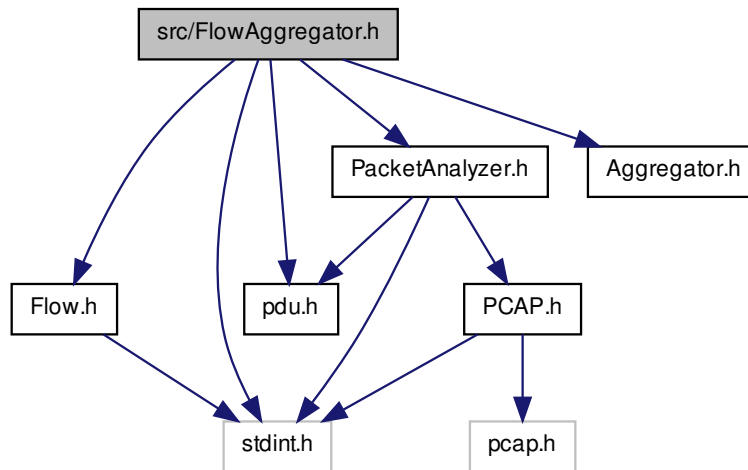
6.16.2.9 **TUDPUA\*** UDPUA

**6.17 src/FlowAggregator.h File Reference**

This file contains the prototypes of the [TFlowAggregator](#) class.

```
#include <stdint.h> #include "PacketAnalyzer.h" #include  
"Flow.h" #include "Aggregator.h" #include "pdu.h" Include de-
```

pendency graph for FlowAggregator.h:



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



## Classes

- class [TFlowAggregator](#)  
*Important class that manages the flows.*

## Defines

- #define [FLOWBUFFERSIZE](#) 65536

### 6.17.1 Detailed Description

This file contains the prototypes of the [TFlowAggregator](#) class.

**Copyright**

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

**Date**

Tuesday, December 4, 2012

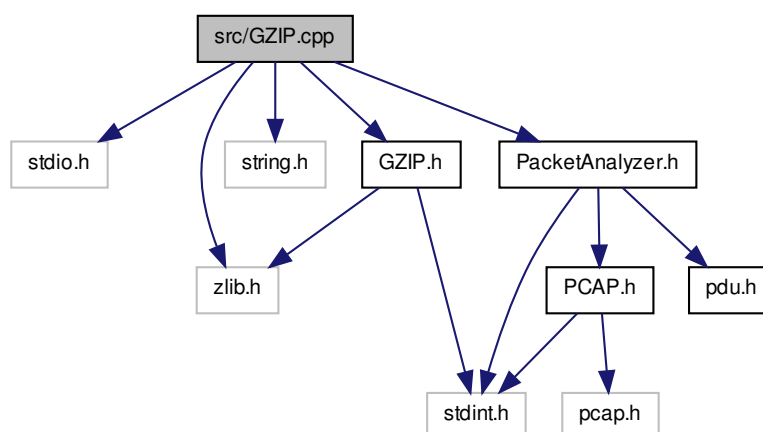
**6.17.2 Define Documentation**

6.17.2.1 `#define FLOWBUFFERSIZE 65536`

**6.18 src/GZIP.cpp File Reference**

This file contains the operators of the [TGZIP](#) class.

```
#include <stdio.h> #include <zlib.h> #include <string.-  
h> #include "GZIP.h" #include "PacketAnalyzer.h" Include depen-  
dency graph for GZIP.cpp:
```



## Defines

- #define [windowBits](#) 15
- #define [ENABLE\\_ZLIB\\_GZIP](#) 32

## Variables

- [TPacketAnalyzer](#) \* [PacketAnalyzer](#)

### 6.18.1 Detailed Description

This file contains the operators of the [TGZIP](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Tuesday, March 12, 2013

### 6.18.2 Define Documentation

6.18.2.1 #define [ENABLE\\_ZLIB\\_GZIP](#) 32

6.18.2.2 #define [windowBits](#) 15

### 6.18.3 Variable Documentation

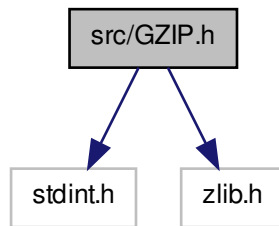
6.18.3.1 [TPacketAnalyzer](#)\* [PacketAnalyzer](#)

## 6.19 src/GZIP.h File Reference

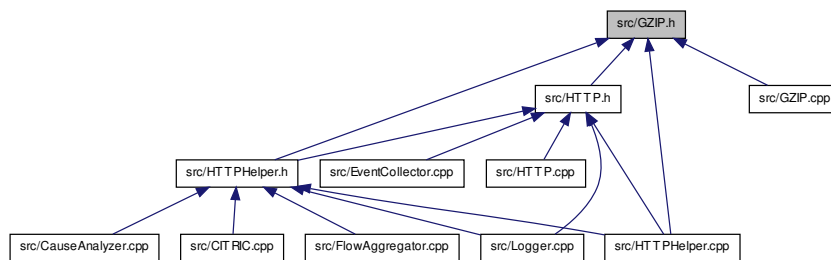
This file contains the prototypes of the [TGZIP](#) class.

```
#include <stdint.h> #include <zlib.h>
```

Include dependency graph for GZIP.h:



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



## Classes

- class [TGZIP](#)  
*Class for deflating gzip data. Wraps around zlib.*

## Defines

- #define [GZIP\\_BUFFERSIZE](#) 32768

### 6.19.1 Detailed Description

This file contains the prototypes of the [TGZIP](#) class.

**Copyright**

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

**Date**

Thursday, February 28, 2013

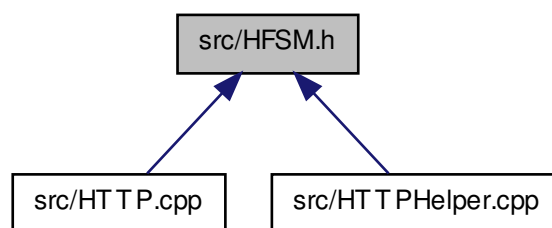
**6.19.2 Define Documentation**

6.19.2.1 `#define GZIP_BUFFERSIZE 32768`

**6.20 src/HFSM.h File Reference**

This file contains HFSM-state definitions.

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

**Defines**

- `#define HFSM1_IDLE 0`
- `#define HFSM1_NO_HTTP 1`



- #define [HFSM1\\_HEADER](#) 2
- #define [HFSM1\\_BODY](#) 3
- #define [HFSM1\\_PARSED](#) 4
- #define [HFSM1\\_CHUNK](#) 5
- #define [HFSM2\\_IDLE](#) 0
- #define [HFSM2\\_NEWLINERECEIVED](#) 1
- #define [HFSM2\\_CONTENT](#) 2
- #define [HFSM2\\_CONTENTTYPE](#) 3
- #define [HFSM2\\_ENCODING](#) 4
- #define [HFSM2\\_CHUNKED](#) 5
- #define [HFSM2\\_CONTENTLENGTH](#) 6
- #define [HFSM2\\_WHITELINE](#) 7
- #define [HFSM2\\_CHAR\\_RECEIVED](#) 8
- #define [HFSM2\\_DOT\\_RECEIVED](#) 9
- #define [HFSM2\\_OVERFLOW](#) 10
- #define [HFSM2\\_MOVED](#) 11
- #define [HFSM2\\_](#)

### 6.20.1 Detailed Description

This file contains HFSM-state definitions.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Wednesday, November 28, 2012

### 6.20.2 Define Documentation

6.20.2.1 #define [HFSM1\\_BODY](#) 3

6.20.2.2 #define [HFSM1\\_CHUNK](#) 5

6.20.2.3 #define [HFSM1\\_HEADER](#) 2

6.20.2.4 `#define HFSM1_IDLE 0`

6.20.2.5 `#define HFSM1_NO_HTTP 1`

6.20.2.6 `#define HFSM1_PARSED 4`

6.20.2.7 `#define HFSM2_`

6.20.2.8 `#define HFSM2_CHAR_RECEIVED 8`

6.20.2.9 `#define HFSM2_CHUNKED 5`

6.20.2.10 `#define HFSM2_CONTENT 2`

6.20.2.11 `#define HFSM2_CONTENTLENGTH 6`

6.20.2.12 `#define HFSM2_CONTENTTYPE 3`

6.20.2.13 `#define HFSM2_DOT_RECEIVED 9`

6.20.2.14 `#define HFSM2_ENCODING 4`

6.20.2.15 `#define HFSM2_IDLE 0`

6.20.2.16 `#define HFSM2_MOVED 11`

6.20.2.17 `#define HFSM2_NEWLINERECEIVED 1`

6.20.2.18 `#define HFSM2_OVERFLOW 10`

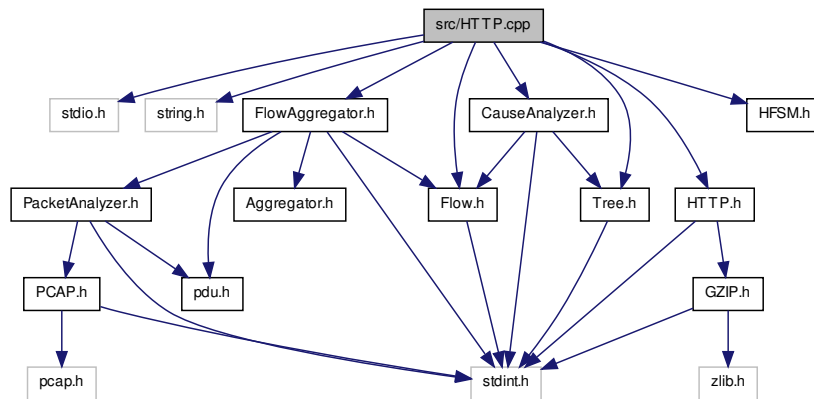
6.20.2.19 `#define HFSM2_WHITELINE 7`

## 6.21 src/HTTP.cpp File Reference

This file contains the operators of the THTTTP class.

```
#include <stdio.h>    #include <string.h>    #include "HTT-  
P.h" #include "HFSM.h" #include "Flow.h" #include "Flow-  
Aggregator.h" #include "CauseAnalyzer.h" #include "Tree.-
```

h" Include dependency graph for HTTP.cpp:



## Variables

- [TFlow Flow](#) [`FLOWBUFFERSIZE`]
- [TTree Tree](#) [`TREE_BUFFER_SIZE`]

### 6.21.1 Detailed Description

This file contains the operators of the THTTTP class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Tuesday, March 5, 2013



## Classes

- class [THTTP](#)

*Storage class for HTTP-events. Every HTTP-flow or HTTPS-flow that can potentially trigger new traffi by embedded URLs has an object of this class.*

## Defines

- #define [HTTPSTATUS\\_UNDEFINED](#) 0
- #define [HTTPSTATUS\\_WAITINGFORSEND](#) 1
- #define [HTTPSTATUS\\_SENT](#) 2
- #define [HTTPSTATUS\\_RECEIVED](#) 3
- #define [REFSTAT\\_UNDEFINED](#) 0
- #define [REFSTAT\\_REF](#) 1
- #define [REFSTAT\\_NOREF](#) 2
- #define [REFSTAT\\_GETSEEN](#) 3

### 6.22.1 Detailed Description

This file contains the prototypes of the [THTTP](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Tuesday, March 5, 2013

### 6.22.2 Define Documentation

6.22.2.1 #define [HTTPSTATUS\\_RECEIVED](#) 3

6.22.2.2 #define [HTTPSTATUS\\_SENT](#) 2

6.22.2.3 #define [HTTPSTATUS\\_UNDEFINED](#) 0

6.22.2.4 `#define HTTPSTATUS_WAITINGFORSEND 1`

6.22.2.5 `#define REFSTAT_GETSEEN 3`

6.22.2.6 `#define REFSTAT_NOREF 2`

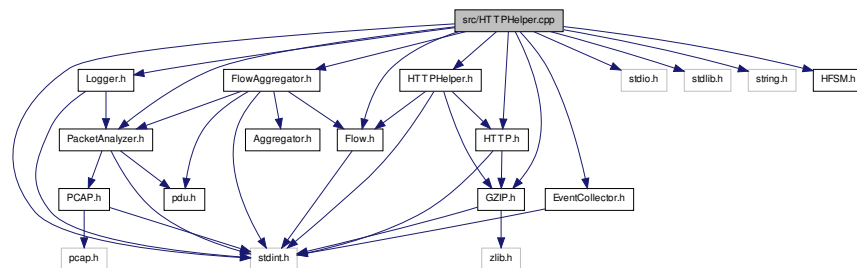
6.22.2.7 `#define REFSTAT_REF 1`

6.22.2.8 `#define REFSTAT_UNDEFINED 0`

## 6.23 src/HTTPHelper.cpp File Reference

This file contains the operators of the THTTPAggregator class.

```
#include <stdint.h> #include <stdio.h> #include <stdlib.-
h> #include <string.h> #include "FlowAggregator.h" #include
"Flow.h" #include "HTTPHelper.h" #include "PacketAnalyzer.-
h" #include "EventCollector.h" #include "HTTP.h" #include
"HFSM.h" #include "GZIP.h" #include "Logger.h" Include depen-
dency graph for HTTPHelper.cpp:
```



## Variables

- TFlowAggregator \* FlowAggregator
- TPCAP \* PCAP
- TPacketAnalyzer \* PacketAnalyzer
- TFlow Flow [FLOWBUFFERSIZE]
- TEventCollector \* EventCollector
- TLogger \* Logger
- THTTP HTTP [HTTP\_BUFFER\_SIZE]
- const char \* PatternHTTP = "HTTP"
- const char \* PatternContentTypeText = "Content-Type:text "
- const char \* PatternContentTypeJava = "Content-Type:application/x-java "
- const char \* PatternContentTypeJS = "Content-Type:application/javascript "
- const char \* PatternContentTypeFlash = "Content-Type:application/x-shoc "

- const char \* [PatternContentTypeJSON](#) = "Content-Type:application/json "
- const char \* [PatternEncodingGZIP](#) = "Encoding:gzip"
- const char \* [PatternTransfer](#) = "Transfer-Encoding:chunked"
- const char \* [PatternLocation](#) = "Location:"
- const char \* [PatternWhiteLine](#) = "\r\n"
- const char \* [Patternhttp](#) = "http"
- const char \* [TLD](#) [26]

### 6.23.1 Detailed Description

This file contains the operators of the THTTPAggregator class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Tuesday, March 12, 2013

### 6.23.2 Variable Documentation

6.23.2.1 TEventCollector\* EventCollector

6.23.2.2 TFlow Flow[FLOWBUFFERSIZE]

6.23.2.3 TFlowAggregator\* FlowAggregator

6.23.2.4 THTTP HTTP[HTTP\_BUFFER\_SIZE]

6.23.2.5 TLogger\* Logger

6.23.2.6 TPacketAnalyzer\* PacketAnalyzer

6.23.2.7 const char\* PatternContentTypeFlash = "Content-Type:application/x-shoc "

6.23.2.8 const char\* PatternContentTypeJava = "Content-Type:application/x-java "

6.23.2.9 `const char* PatternContentTypeJS = "Content-Type:application/javascript "`

6.23.2.10 `const char* PatternContentTypeJSON = "Content-Type:application/json "`

6.23.2.11 `const char* PatternContentTypeText = "Content-Type:text "`

6.23.2.12 `const char* PatternEncodingGZIP = "Encoding:gzip"`

6.23.2.13 `const char* PatternHTTP = "HTTP"`

6.23.2.14 `const char* Patternhttp = "http"`

6.23.2.15 `const char* PatternLocation = "Location:"`

6.23.2.16 `const char* PatternTransfer = "Transfer-Encoding:chunked"`

6.23.2.17 `const char* PatternWhiteLine = "\r\n"`

6.23.2.18 **TPCAP\* PCAP**

6.23.2.19 `const char* TLD[26]`

**Initial value:**

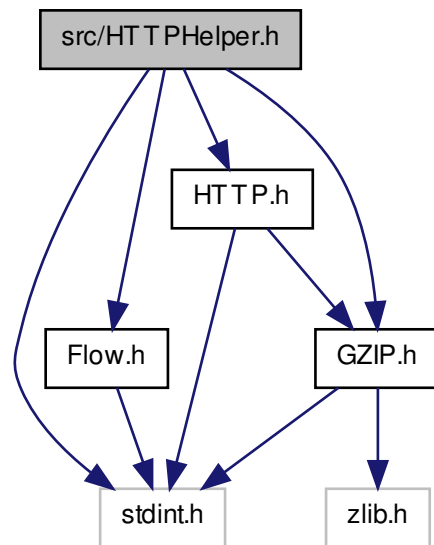
```
{
  ".ac.ad.ae.aero.af.ag.ai.al.am.an.ao.aq.ar.arpa.as.asia.at.au.aw.ax.az",
  ".ba.bb.bd.be.bf.bg.bh.bi.biz.bj.bl.bm.bn.bo.bq.br.bs.bt.bv.bw.by.bz",
  ".ca.cat.cc.cd.cf.cg.ch.ci.ck.cl.cm.cn.co.com.coop.cr.cu.cv.cw.cx.cy.cz",
  ".de.dj.dk.dm.do.dz",
  ".ec.edu.ee.eg.eh.er.es.et.eu",
  ".fi.fj.fk.fm.fo.fr",
  ".ga.gb.gd.ge.gf.gg.gh.gi.gl.gm.gn.gov.gp.gq.gr.gs.gt.gu.gw.gy",
  ".hk.hm.hn.hr.ht.hu",
  ".id.ie.il.im.in.info.int.io.iq.ir.is.it",
  ".je.jm.jo.jobs.jp",
  ".ke.kg.kh.ki.km.kn.kp.kr.kw.ky.kz",
  ".la.lb.lc.li.lk.lr.ls.lt.lu.lv.ly",
  "",
  ".ma.mc.md.me.mf.mg.mh.mil.mk.ml.mm.mn.mo.mobi.mp.mq.mr.ms.mt.mu.museum.mv.mw.mx.my.mz",
  ".na.name.nc.ne.net.nf.ng.ni.nl.no.np.nr.nu.nz",
  ".om.org",
  ".pa.pe.pf.pg.ph.pk.pl.pm.pn.post.pr.pro.ps.pt.pw.py",
  ".qa",
  ".re.ro.rs.ru.rw",
  ".sa.sb.sc.sd.se.sg.sh.si.sj.sk.sl.sm.sn.so.sr.ss.st.su.sv.sx.sy.sz",
  ".tc.td.tel.tf.tg.th.tj.tk.tl.tm.tn.to.tp.tr.travel.tt.tv.tw.tz",
  ".ua.ug.uk.um.us.uy.uz",
  ".va.vc.ve.vg.vi.vn.vu",
  ".wf.ws",
  ".xxx.xn--",
  ".ye.yt",
  ".za.zm.zw"
}
```



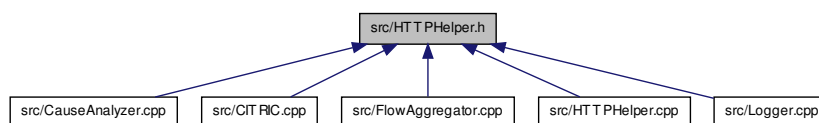
## 6.24 src/HTTPHelper.h File Reference

This file contains the prototypes of the [THTTPHelper](#) class.

```
#include <stdint.h> #include "Flow.h" #include "HTTP.h" ×  
#include "GZIP.h" Include dependency graph for HTTPHelper.h:
```



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



### Classes

- class [THTTPHelper](#)

*Class that aggregates HTTP traffic. It aggregates information that predicts the cause of new traffic flows. [THTTPHelper](#) creates a static array of `HTTP_BUFFER_SIZE` (65536)*

*HTTP-objects. The ClusterAggregator uses this class to check for potential HTTP-parent flows. [TFlowAggregator](#) updates the information of this class.*

## Defines

- #define [HTTP\\_BUFFER\\_SIZE](#) 65536
- #define [TOTAL\\_GZIP](#) 2000
- #define [BINARYCONTENT](#) 0

### 6.24.1 Detailed Description

This file contains the prototypes of the [THTTPHelper](#) class.

## Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

## Author

Pieter Burghouwt

## Version

Revision 2.0

## Date

Tuesday, March 5, 2013

### 6.24.2 Define Documentation

6.24.2.1 #define [BINARYCONTENT](#) 0

6.24.2.2 #define [HTTP\\_BUFFER\\_SIZE](#) 65536

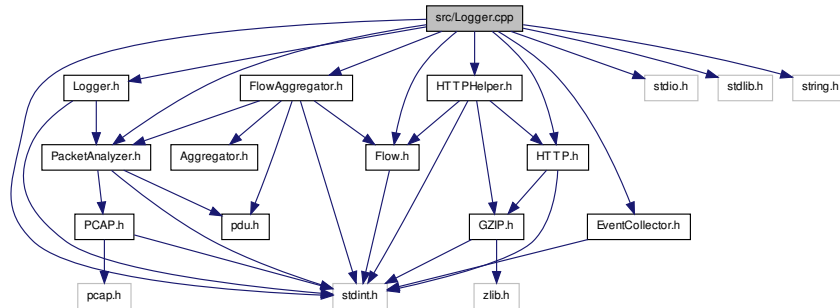
6.24.2.3 #define [TOTAL\\_GZIP](#) 2000

## 6.25 src/Logger.cpp File Reference

This file contains the operators of the [TLogger](#) class.

```
#include <stdint.h> #include <stdio.h> #include <stdlib.-  
h> #include <string.h> #include "FlowAggregator.h" #include
```

```
"Flow.h" #include "HTTPHelper.h" #include "PacketAnalyzer.h"
#include "EventCollector.h" #include "HTTP.h" #include
"Logger.h" Include dependency graph for Logger.cpp:
```



## Variables

- `TFlowAggregator` \* `FlowAggregator`
- `TPCAP` \* `PCAP`
- `TPacketAnalyzer` \* `PacketAnalyzer`
- `TFlow` `Flow` [`FLOWBUFFERSIZE`]
- `TEventCollector` \* `EventCollector`
- `THTTP` \* `HTTP`
- `const char` \* `Severity` [8] = {"EMERGENCY", "**ALERT**", "**CRITICAL**", "**ERROR**", "**WARNING**", "**NOTICE**", "**INFORMATIONAL**", "**DEBUG**"}
- `const char` \* `Facility` [7] = {"PACKET\_ANALYZER", "FLOW\_AGGREGATOR", "DNS\_HELPER", "HTTP\_HELPER", "EVENT\_COLLECTOR", "CAUSE\_ANALYZER", "MAIN"}
- `const char` \* `Protocol` [7] = {"ICMP", "**TCP**", "**UDP**", "**HTTP**", "HTTPS", "**DNS**", "UNKNOWN"}
- `const char` \* `CauseDesc` [13] = {"CAUSE\_UNKNOWN", "**CAUSE\_SERVER**", "**CAUSE\_DNS**", "**CAUSE\_HTTP\_URL**", "**CAUSE\_HTTP\_SOFTURL**", "**CAUSE\_HTTP\_REFERER**", "**CAUSE\_HTTP\_GEN**", "**CAUSE\_HTTPS\_GEN**", "**CAUSE\_USER**", "**CAUSE\_PROTODNS**", "**CAUSE\_WHITELIST**", "**CAUSE\_ALREADYOPEN**", "**CAUSE\_DNS\_REPEAT**"}

### 6.25.1 Detailed Description

This file contains the operators of the `TLogger` class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

**Date**

Wednesday, December 5, 2012

**6.25.2 Variable Documentation**

**6.25.2.1** `const char* CauseDesc[13] = { "CAUSE_UNKNOWN", "CAUSE_SERVER",  
"CAUSE_DNS", "CAUSE_HTTP_URL", "CAUSE_HTTP_SOFTURL",  
"CAUSE_HTTP_REFERER", "CAUSE_HTTP_GEN", "CAUSE_HTTPS_G-  
EN", "CAUSE_USER", "CAUSE_PROTODNS", "CAUSE_WHITELIST",  
"CAUSE_ALREADYOPEN", "CAUSE_DNS_REPEAT" }`

**6.25.2.2** `TEventCollector* EventCollector`

**6.25.2.3** `const char* Facility[7] = { "PACKET_ANALYZER", "FLOW_AGGREGATOR",  
"DNS_HELPER", "HTTP_HELPER", "EVENT_COLLECTOR", "CAUSE_ANALYZER",  
"MAIN" }`

**6.25.2.4** `TFlow Flow[FLOWBUFFERSIZE]`

**6.25.2.5** `TFlowAggregator* FlowAggregator`

**6.25.2.6** `THTTP* HTTP`

**6.25.2.7** `TPacketAnalyzer* PacketAnalyzer`

**6.25.2.8** `TPCAP* PCAP`

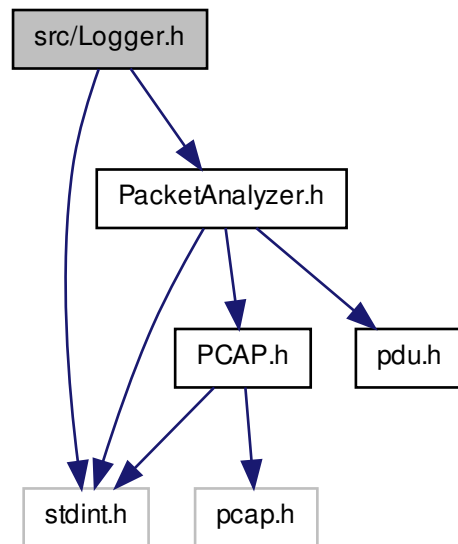
**6.25.2.9** `const char* Protocol[7] = { "ICMP", "TCP", "UDP", "HTTP", "HTTPS", "DNS",  
"UNKNOWN" }`

**6.25.2.10** `const char* Severity[8] = { "EMERGENCY", "ALERT", "CRITICAL", "ERROR",  
"WARNING", "NOTICE", "INFORMATIONAL", "DEBUG" }`

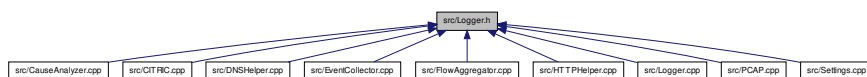
**6.26 src/Logger.h File Reference**

This file contains the prototypes of the [TLogger](#) class.

```
#include <stdint.h> #include "PacketAnalyzer.h" Include dependency graph for Logger.h:
```



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



## Classes

- class `TLogger`

*Class that aggregates log messages in flat file format in RAM.*

## Defines

- `#define LOGSIZE 10000000`
- `#define DEBUG 7`
- `#define INFORMATIONAL 6`

- #define NOTICE 5
- #define WARNING 4
- #define ERROR 3
- #define CRITICAL 2
- #define ALERT 1
- #define EMERGENCY 0
- #define FAC\_PACKET 0
- #define FAC\_FLOW 1
- #define FAC\_DNS 2
- #define FAC\_HTTP 3
- #define FAC\_EVENT 4
- #define FAC\_CAUSE 5
- #define FAC\_MAIN 6

### 6.26.1 Detailed Description

This file contains the prototypes of the [TLogger](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Wednesday, December 5, 2012

### 6.26.2 Define Documentation

6.26.2.1 #define ALERT 1

6.26.2.2 #define CRITICAL 2

6.26.2.3 #define DEBUG 7

6.26.2.4 #define EMERGENCY 0

6.26.2.5 #define ERROR 3

6.26.2.6 `#define FAC_CAUSE 5`

6.26.2.7 `#define FAC_DNS 2`

6.26.2.8 `#define FAC_EVENT 4`

6.26.2.9 `#define FAC_FLOW 1`

6.26.2.10 `#define FAC_HTTP 3`

6.26.2.11 `#define FAC_MAIN 6`

6.26.2.12 `#define FAC_PACKET 0`

6.26.2.13 `#define INFORMATIONAL 6`

6.26.2.14 `#define LOGSIZE 10000000`

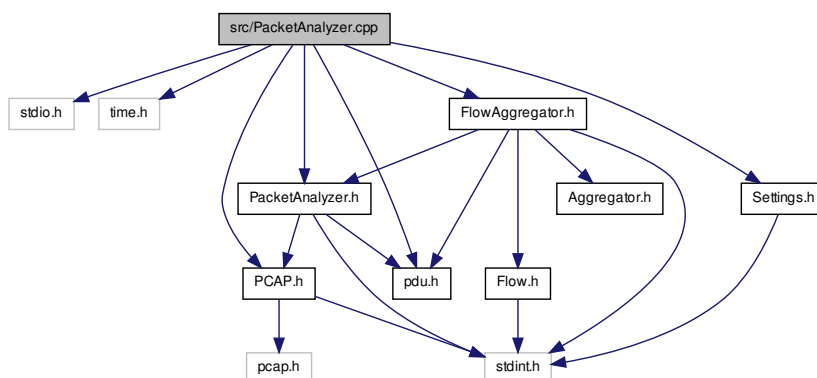
6.26.2.15 `#define NOTICE 5`

6.26.2.16 `#define WARNING 4`

## 6.27 src/PacketAnalyzer.cpp File Reference

This file contains the operators of the [TPacketAnalyzer](#) class.

```
#include <stdio.h> #include <time.h> #include "pdu.h" ×
#include "PacketAnalyzer.h" #include "PCAP.h" #include "-
FlowAggregator.h" #include "Settings.h" Include dependency graph
for PacketAnalyzer.cpp:
```



## Variables

- [TPCAP](#) \* [PCAP](#)
- [TFlowAggregator](#) \* [FlowAggregator](#)
- [TSettings](#) \* [Settings](#)

### 6.27.1 Detailed Description

This file contains the operators of the [TPacketAnalyzer](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Tuesday, March 12, 2013

### 6.27.2 Variable Documentation

#### 6.27.2.1 [TFlowAggregator](#)\* [FlowAggregator](#)

#### 6.27.2.2 [TPCAP](#)\* [PCAP](#)

#### 6.27.2.3 [TSettings](#)\* [Settings](#)

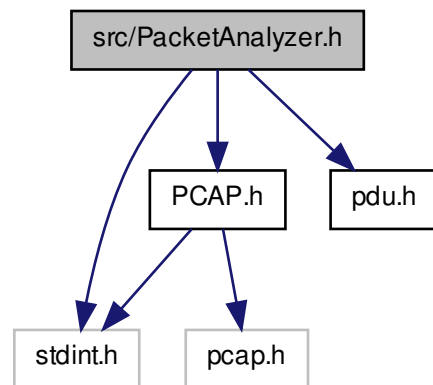
## 6.28 src/PacketAnalyzer.h File Reference

This file contains the prototypes of the [TPacketAnalyzer](#) class.



```
#include <stdint.h> #include "PCAP.h" #include "pdu.h" ×
```

Include dependency graph for PacketAnalyzer.h:



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



## Classes

- class `TPacketAnalyzer`  
*Class for analyzing an arrived packet.*

### 6.28.1 Detailed Description

This file contains the prototypes of the `TPacketAnalyzer` class.

## Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

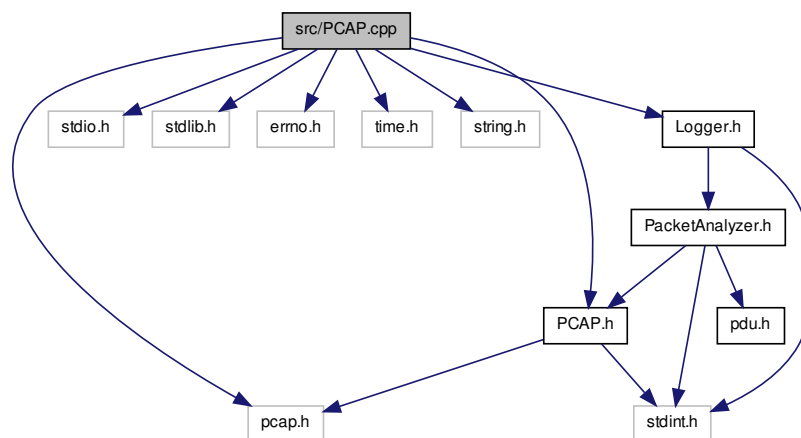
**Date**

Tuesday, March 12, 2013

## 6.29 src/PCAP.cpp File Reference

This file contains the operators of the [TPCAP](#) class.

```
#include <pcap.h> #include <stdio.h> #include <stdlib.-  
h> #include <errno.h> #include <time.h> #include <string.-  
h> #include "PCAP.h" #include "Logger.h" Include dependency graph  
for PCAP.cpp:
```

**Variables**

- [TLogger](#) \* [Logger](#)

### 6.29.1 Detailed Description

This file contains the operators of the [TPCAP](#) class.

**Copyright**

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

**Author**

Pieter Burghouwt

**Version**

Revision 2.0

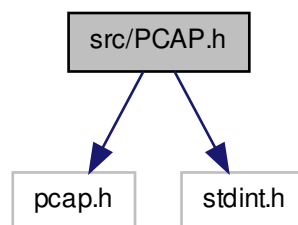
**Date**

Tuesday, November 22, 2011

**6.29.2 Variable Documentation****6.29.2.1 TLogger\* Logger****6.30 src/PCAP.h File Reference**

This file contains the prototypes of the [TPCAP](#) class.

`#include <pcap.h> #include <stdint.h>` Include dependency graph for PCAP.h:



[illegible]

- class TPCAP

### 6.30.1 Detailed Description

## Copyright

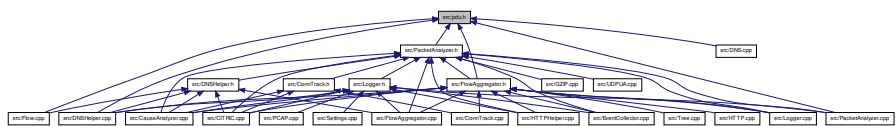
**Author**

## Version

Date \_\_\_\_\_

### 6.31 src/pdu.h File Reference

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



## Defines

- #define [ETHERNET\\_LENGTH](#) 14
- #define [IP\\_MIN\\_LENGTH](#) 20
- #define [UDP\\_LENGTH](#) 8
- #define [ICMP\\_LENGTH](#) 4
- #define [ETH\\_PROT\\_OFFSET](#) 12
- #define [IP\\_VERSION\\_OFFSET](#) 0
- #define [IP\\_LENGTH\\_OFFSET](#) 0
- #define [IP\\_PROT\\_OFFSET](#) 9
- #define [IP\\_SOURCE\\_OFFSET](#) 12
- #define [IP\\_DEST\\_OFFSET](#) 16
- #define [UDP\\_LENGTH](#) 8
- #define [TCP\\_LENGTH\\_OFFSET](#) 12
- #define [TCP\\_SEQ\\_OFFSET](#) 4
- #define [TCP\\_ACK\\_OFFSET](#) 8
- #define [TCP\\_FLAG\\_OFFSET](#) 13
- #define [SOURCE\\_PORT\\_OFFSET](#) 0
- #define [DEST\\_PORT\\_OFFSET](#) 2
- #define [ICMP\\_TYPE\\_OFFSET](#) 0
- #define [ICMP\\_CODE\\_OFFSET](#) 1
- #define [ICMP\\_ID\\_OFFSET](#) 4
- #define [ICMP\\_SEQ\\_OFFSET](#) 6
- #define [TCP](#) 6
- #define [UDP](#) 17
- #define [ICMP](#) 1
- #define [OTHER](#) 255
- #define [DNS\\_IDENTIFICATION](#) 0
- #define [DNS\\_OPCODE\\_OFFSET](#) 2
- #define [DNS\\_RCODE\\_OFFSET](#) 3
- #define [DNS\\_NUMBER\\_OF\\_QUESTIONS\\_OFFSET](#) 4
- #define [DNS\\_NUMBER\\_OF\\_ANSWERS\\_OFFSET](#) 6
- #define [DNS\\_QUESTIONS\\_OFFSET](#) 12

### 6.31.1 Detailed Description

This file contains PDU-fields, places and values.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

**Version**

Revision 2.0

**Date**

Saturday, March 9, 2013

**6.31.2 Define Documentation**6.31.2.1 `#define DEST_PORT_OFFSET 2`6.31.2.2 `#define DNS_IDENTIFICATION 0`6.31.2.3 `#define DNS_NUMBER_OF_ANSWERS_OFFSET 6`6.31.2.4 `#define DNS_NUMBER_OF_QUESTIONS_OFFSET 4`6.31.2.5 `#define DNS_OPCODE_OFFSET 2`6.31.2.6 `#define DNS_QUESTIONS_OFFSET 12`6.31.2.7 `#define DNS_RCODE_OFFSET 3`6.31.2.8 `#define ETH_PROT_OFFSET 12`6.31.2.9 `#define ETHERNET_LENGTH 14`6.31.2.10 `#define ICMP 1`6.31.2.11 `#define ICMP_CODE_OFFSET 1`6.31.2.12 `#define ICMP_ID_OFFSET 4`6.31.2.13 `#define ICMP_LENGTH 4`6.31.2.14 `#define ICMP_SEQ_OFFSET 6`6.31.2.15 `#define ICMP_TYPE_OFFSET 0`6.31.2.16 `#define IP_DEST_OFFSET 16`6.31.2.17 `#define IP_LENGTH_OFFSET 0`6.31.2.18 `#define IP_MIN_LENGTH 20`6.31.2.19 `#define IP_PROT_OFFSET 9`

6.31.2.20 `#define IP_SOURCE_OFFSET 12`

6.31.2.21 `#define IP_VERSION_OFFSET 0`

6.31.2.22 `#define OTHER 255`

6.31.2.23 `#define SOURCE_PORT_OFFSET 0`

6.31.2.24 `#define TCP 6`

6.31.2.25 `#define TCP_ACK_OFFSET 8`

6.31.2.26 `#define TCP_FLAG_OFFSET 13`

6.31.2.27 `#define TCP_LENGTH_OFFSET 12`

6.31.2.28 `#define TCP_SEQ_OFFSET 4`

6.31.2.29 `#define UDP 17`

6.31.2.30 `#define UDP_LENGTH 8`

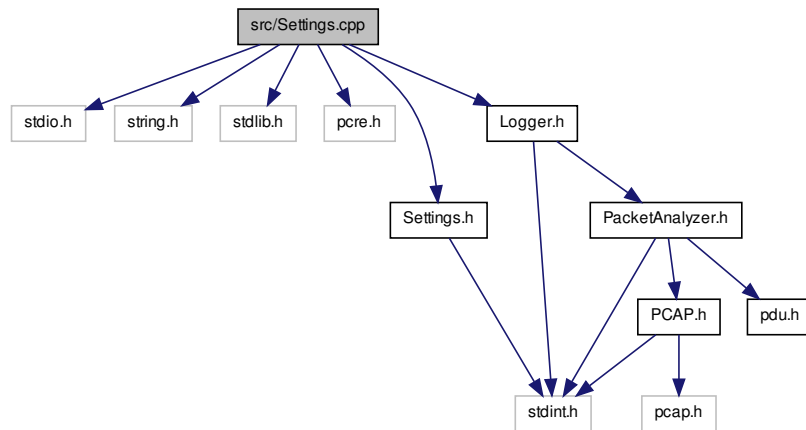
6.31.2.31 `#define UDP_LENGTH 8`

## 6.32 src/Settings.cpp File Reference

This file contains the operators of the [TSettings](#) class.

```
#include <stdio.h> #include <string.h> #include <stdlib.-  
h> #include <pcres.h> #include "Settings.h" #include "-
```

Logger.h" Include dependency graph for Settings.cpp:



## Variables

- [TLogger](#) \* [Logger](#)

### 6.32.1 Detailed Description

This file contains the operators of the [TSettings](#) class.

## Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

## Author

Pieter Burghouwt

## Version

Revision 2.0

## Date

Saturday, December 22, 2012



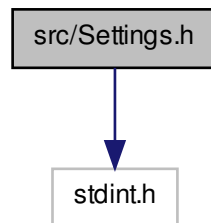
### 6.32.2 Variable Documentation

#### 6.32.2.1 TLogger\* Logger

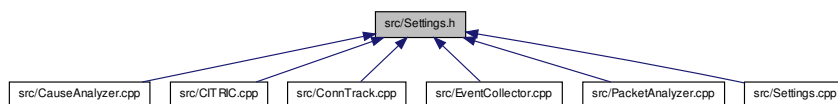
## 6.33 src/Settings.h File Reference

This file contains the prototypes of the [TSettings](#) class.

`#include <stdint.h>` Include dependency graph for Settings.h:



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



### Classes

- class [TSettings](#)

### Defines

- `#define` [SETTINGSSIZE](#) 13
- `#define` [DELTA\\_T\\_DNS Settings->DefinedValues\[0\]](#)
- `#define` [DELTA\\_T\\_DNS\\_DEL Settings->DefinedValues\[1\]](#)
- `#define` [DELTA\\_T\\_DNS\\_RPT Settings->DefinedValues\[2\]](#)
- `#define` [DELTA\\_T\\_URL Settings->DefinedValues\[3\]](#)

- #define [DELTA\\_T\\_URL\\_DEL Settings](#)->DefinedValues[4]
- #define [DELTA\\_T\\_HTTP Settings](#)->DefinedValues[5]
- #define [DELTA\\_T\\_HTTPS Settings](#)->DefinedValues[6]
- #define [DELTA\\_T\\_USER Settings](#)->DefinedValues[7]
- #define [DELTA\\_T\\_UTREE Settings](#)->DefinedValues[8]
- #define [IPS\\_ENABLE Settings](#)->DefinedValues[9]
- #define [DNS\\_PORTPATCH Settings](#)->DefinedValues[10]
- #define [IDL\\_MAX\\_TOKENS Settings](#)->DefinedValues[11]
- #define [IDL\\_MAX\\_LENGTH Settings](#)->DefinedValues[12]

### 6.33.1 Detailed Description

This file contains the prototypes of the [TSettings](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Saturday, December 22, 2012

### 6.33.2 Define Documentation

- 6.33.2.1 #define [DELTA\\_T\\_DNS Settings](#)->DefinedValues[0]
- 6.33.2.2 #define [DELTA\\_T\\_DNS\\_DEL Settings](#)->DefinedValues[1]
- 6.33.2.3 #define [DELTA\\_T\\_DNS\\_RPT Settings](#)->DefinedValues[2]
- 6.33.2.4 #define [DELTA\\_T\\_HTTP Settings](#)->DefinedValues[5]
- 6.33.2.5 #define [DELTA\\_T\\_HTTPS Settings](#)->DefinedValues[6]
- 6.33.2.6 #define [DELTA\\_T\\_URL Settings](#)->DefinedValues[3]
- 6.33.2.7 #define [DELTA\\_T\\_URL\\_DEL Settings](#)->DefinedValues[4]

6.33.2.8 `#define DELTA_T_USER` Settings->DefinedValues[7]

6.33.2.9 `#define DELTA_T_UTREE` Settings->DefinedValues[8]

6.33.2.10 `#define DNS_PORTPATCH` Settings->DefinedValues[10]

6.33.2.11 `#define IDL_MAX_LENGTH` Settings->DefinedValues[12]

6.33.2.12 `#define IDL_MAX_TOKENS` Settings->DefinedValues[11]

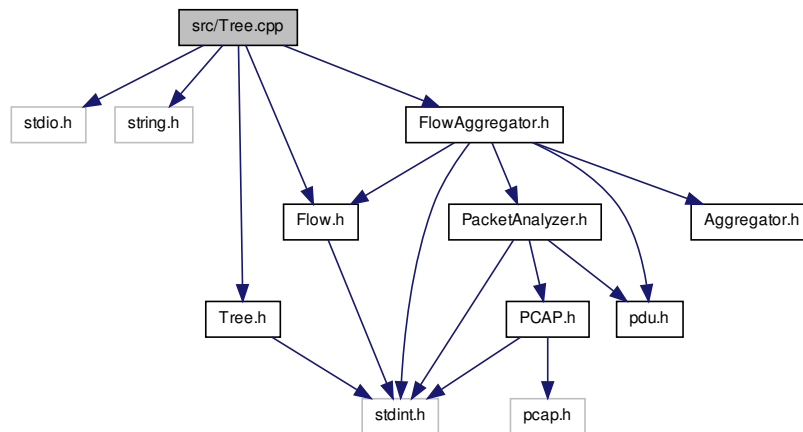
6.33.2.13 `#define IPS_ENABLE` Settings->DefinedValues[9]

6.33.2.14 `#define SETTINGSSIZE` 13

## 6.34 src/Tree.cpp File Reference

This file contains the operators of the [TTree](#) class.

```
#include <stdio.h> #include <string.h> #include "Tree.h"
#include "Flow.h" #include "FlowAggregator.h" Include dependency graph for Tree.cpp:
```



### Variables

- [TFlow Flow](#) [[FLOWBUFFERSIZE](#)]

### 6.34.1 Detailed Description

This file contains the operators of the [TTree](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Friday, March 8, 2013

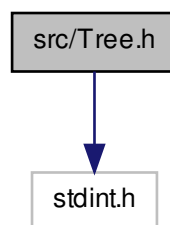
### 6.34.2 Variable Documentation

#### 6.34.2.1 TFlow Flow[FLOWBUFFERSIZE]

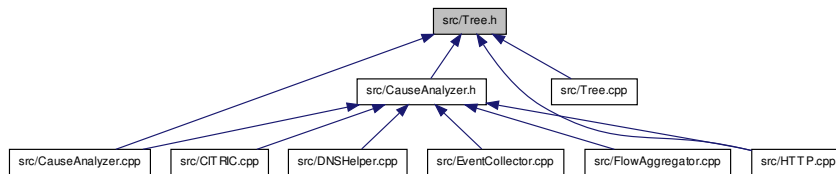
## 6.35 src/Tree.h File Reference

This file contains the prototypes of the [TTree](#) class.

`#include <stdint.h>` Include dependency graph for Tree.h:



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



## Classes

- class [TTree](#)

*Storage class for a tree of network flows with a causal relationship.*

### 6.35.1 Detailed Description

This file contains the prototypes of the [TTree](#) class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Pieter Burghouwt

#### Version

Revision 2.0

#### Date

Friday, March 8, 2013

## 6.36 src/UDPUA.cpp File Reference

This file contains the operators of the [TUDPUA](#) class.

```
#include <stdlib.h> #include <stdio.h> #include <fcntl.-  
h> #include <errno.h> #include <sys/types.h> #include
```



## 6.36.2.2 TPacketAnalyzer\* PacketAnalyzer

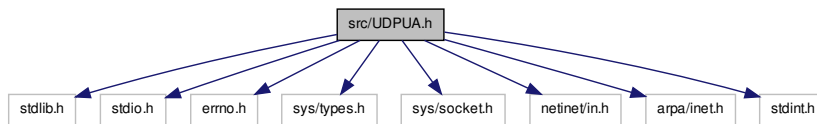
## 6.36.2.3 TPCAP\* PCAP

## 6.37 src/UDPUA.h File Reference

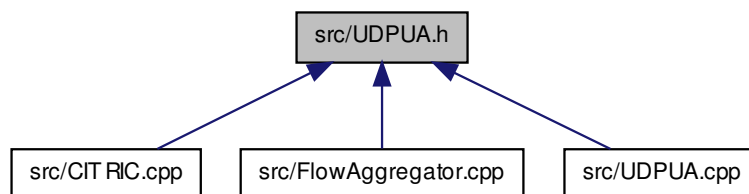
This file contains the prototypes of the UDPUA class.

```
#include <stdlib.h> #include <stdio.h> #include <errno.h> #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <stdint.h>
```

h> Include dependency graph for UDPUA.h:



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



## Classes

- class [TUDPUA](#)

*One object of this class (class should be used as singleton) receives and filters user events, received from UDP with a special agent. Received results can be polled in a Event-loop.*

## Defines

- `#define UDP_Port 1234`

### 6.37.1 Detailed Description

This file contains the prototypes of the UDPUA class.

#### Copyright

Copyright 2012-2013 Delft University of Technology and The Hague University of Applied Sciences. License: LGPL 3+

#### Author

Burghouwt:Pieter

#### Version

Revision 2.0

#### Date

Wednesday, April 4, 2012

### 6.37.2 Define Documentation

#### 6.37.2.1 `#define UDP_Port 1234`