

ESA JPIP server

0.1

Generated by Doxygen 1.8.9.1

Sat Aug 22 2015 12:51:34

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	9
4.1	File List	9
5	Namespace Documentation	11
5.1	data Namespace Reference	11
5.1.1	Detailed Description	12
5.1.2	Typedef Documentation	12
5.1.2.1	FastFile	12
5.1.2.2	File	12
5.2	http Namespace Reference	12
5.2.1	Detailed Description	13
5.2.2	Function Documentation	13
5.2.2.1	operator<<	13
5.2.2.2	operator>>	13
5.3	ipc Namespace Reference	13
5.3.1	Detailed Description	14
5.3.2	Enumeration Type Documentation	14
5.3.2.1	WaitResult	14
5.4	jpeg2000 Namespace Reference	14
5.4.1	Detailed Description	15
5.4.2	Typedef Documentation	15
5.4.2.1	Size	15
5.5	jpip Namespace Reference	15
5.5.1	Detailed Description	16

5.6	net Namespace Reference	16
5.6.1	Detailed Description	16
6	Class Documentation	17
6.1	net::Address Class Reference	17
6.1.1	Detailed Description	17
6.1.2	Constructor & Destructor Documentation	17
6.1.2.1	Address	17
6.1.2.2	~Address	18
6.1.3	Member Function Documentation	18
6.1.3.1	GetSize	18
6.1.3.2	GetSockAddr	18
6.2	AppConfig Class Reference	18
6.2.1	Detailed Description	19
6.2.2	Constructor & Destructor Documentation	19
6.2.2.1	AppConfig	19
6.2.2.2	~AppConfig	20
6.2.3	Member Function Documentation	20
6.2.3.1	address	20
6.2.3.2	cache_max_time	20
6.2.3.3	caching_folder	20
6.2.3.4	com_time_out	20
6.2.3.5	images_folder	20
6.2.3.6	Load	20
6.2.3.7	log_requests	20
6.2.3.8	logging	21
6.2.3.9	logging_folder	21
6.2.3.10	max_chunk_size	21
6.2.3.11	max_connections	21
6.2.3.12	port	21
6.2.4	Friends And Related Function Documentation	21
6.2.4.1	operator<<	21
6.2.5	Member Data Documentation	21
6.2.5.1	address_	21
6.2.5.2	cache_max_time_	21
6.2.5.3	caching_folder_	21
6.2.5.4	com_time_out_	21
6.2.5.5	images_folder_	22
6.2.5.6	log_requests_	22
6.2.5.7	logging_	22

6.2.5.8	logging_folder_	22
6.2.5.9	max_chunk_size_	22
6.2.5.10	max_connections_	22
6.2.5.11	port_	22
6.3	AppInfo Class Reference	22
6.3.1	Detailed Description	24
6.3.2	Constructor & Destructor Documentation	24
6.3.2.1	AppInfo	24
6.3.2.2	~AppInfo	24
6.3.3	Member Function Documentation	24
6.3.3.1	available_memory	24
6.3.3.2	child_memory	24
6.3.3.3	child_time	24
6.3.3.4	father_memory	24
6.3.3.5	GetProcStat	24
6.3.3.6	GetProcStat_	24
6.3.3.7	Init	25
6.3.3.8	is_running	25
6.3.3.9	num_threads	25
6.3.3.10	operator->	25
6.3.3.11	time	25
6.3.3.12	Update	25
6.3.4	Friends And Related Function Documentation	25
6.3.4.1	operator<<	25
6.3.5	Member Data Documentation	25
6.3.5.1	available_memory_	25
6.3.5.2	child_memory_	25
6.3.5.3	child_time_	25
6.3.5.4	data_ptr	26
6.3.5.5	father_memory_	26
6.3.5.6	is_running_	26
6.3.5.7	lock_file	26
6.3.5.8	num_threads_	26
6.3.5.9	shmid	26
6.3.5.10	time_	26
6.4	ArgsParser Class Reference	26
6.4.1	Detailed Description	27
6.4.2	Constructor & Destructor Documentation	27
6.4.2.1	ArgsParser	27
6.4.3	Member Function Documentation	27

6.4.3.1	Parse	27
6.4.4	Member Data Documentation	27
6.4.4.1	app_info	27
6.4.4.2	cfg	27
6.5	base Struct Reference	28
6.5.1	Detailed Description	28
6.5.2	Member Function Documentation	28
6.5.2.1	copy	28
6.5.2.2	copy	28
6.5.2.3	copy	28
6.5.2.4	to_string	28
6.6	data::BaseFile< IO > Class Template Reference	29
6.6.1	Detailed Description	30
6.6.2	Member Typedef Documentation	30
6.6.2.1	Ptr	30
6.6.3	Constructor & Destructor Documentation	31
6.6.3.1	BaseFile	31
6.6.3.2	~BaseFile	31
6.6.4	Member Function Documentation	31
6.6.4.1	Close	31
6.6.4.2	Exists	31
6.6.4.3	GetDescriptor	31
6.6.4.4	GetOffset	31
6.6.4.5	GetSize	31
6.6.4.6	IsEOF	31
6.6.4.7	IsValid	31
6.6.4.8	Open	32
6.6.4.9	Open	33
6.6.4.10	Open	33
6.6.4.11	OpenForReading	33
6.6.4.12	OpenForReading	33
6.6.4.13	OpenForReading	33
6.6.4.14	OpenForWriting	33
6.6.4.15	OpenForWriting	33
6.6.4.16	OpenForWriting	33
6.6.4.17	operator bool	34
6.6.4.18	Read	34
6.6.4.19	ReadByte	34
6.6.4.20	ReadReverse	34
6.6.4.21	Seek	34

6.6.4.22	Write	35
6.6.4.23	WriteByte	36
6.6.4.24	WriteReverse	36
6.6.5	Member Data Documentation	36
6.6.5.1	file_ptr	36
6.7	data::BaseStream< StreamClass, StreamOperator > Class Template Reference	36
6.7.1	Detailed Description	37
6.7.2	Constructor & Destructor Documentation	37
6.7.2.1	BaseStream	37
6.7.2.2	~BaseStream	38
6.7.3	Member Function Documentation	38
6.7.3.1	Close	38
6.7.3.2	Open	38
6.7.3.3	Open	38
6.7.3.4	operator bool	38
6.7.3.5	operator&	38
6.7.3.6	result	38
6.7.3.7	SerializeBytes	39
6.7.4	Member Data Documentation	39
6.7.4.1	file_	39
6.7.4.2	result_	39
6.8	jpeg::CacheModel Class Reference	39
6.8.1	Detailed Description	40
6.8.2	Constructor & Destructor Documentation	40
6.8.2.1	CacheModel	40
6.8.2.2	CacheModel	40
6.8.2.3	~CacheModel	41
6.8.3	Member Function Documentation	41
6.8.3.1	AddToDataBin	41
6.8.3.2	AddToMetadata	41
6.8.3.3	Clear	41
6.8.3.4	GetCodestream	41
6.8.3.5	GetDataBin	41
6.8.3.6	GetMetadata	42
6.8.3.7	IsFullMetadata	42
6.8.3.8	operator+=	42
6.8.3.9	operator=	42
6.8.3.10	Pack	42
6.8.3.11	SerializeWith	42
6.8.3.12	SetFullMetadata	42

6.8.4	Member Data Documentation	42
6.8.4.1	codestreams	42
6.8.4.2	full_meta	43
6.8.4.3	meta_data	43
6.9	ClientInfo Class Reference	43
6.9.1	Detailed Description	43
6.9.2	Constructor & Destructor Documentation	44
6.9.2.1	ClientInfo	44
6.9.2.2	~ClientInfo	44
6.9.3	Member Function Documentation	44
6.9.3.1	base_id	44
6.9.3.2	bytes_sent	44
6.9.3.3	father_sock	44
6.9.3.4	sock	44
6.9.3.5	time	44
6.9.4	Member Data Documentation	44
6.9.4.1	base_id_	44
6.9.4.2	bytes_sent_	44
6.9.4.3	father_sock_	45
6.9.4.4	sock_	45
6.9.4.5	tm_start	45
6.10	ClientManager Class Reference	45
6.10.1	Detailed Description	45
6.10.2	Constructor & Destructor Documentation	46
6.10.2.1	ClientManager	46
6.10.2.2	~ClientManager	47
6.10.3	Member Function Documentation	47
6.10.3.1	Run	47
6.10.3.2	RunBasic	47
6.10.4	Member Data Documentation	47
6.10.4.1	app_info	47
6.10.4.2	cfg	47
6.10.4.3	index_manager	47
6.11	jpeg::CacheModel::Codestream Class Reference	48
6.11.1	Detailed Description	48
6.11.2	Constructor & Destructor Documentation	49
6.11.2.1	Codestream	49
6.11.2.2	Codestream	49
6.11.3	Member Function Documentation	49
6.11.3.1	AddToMainHeader	49

6.11.3.2	AddToPrecinct	49
6.11.3.3	AddToTileHeader	49
6.11.3.4	GetMainHeader	50
6.11.3.5	GetPrecinct	50
6.11.3.6	GetTileHeader	50
6.11.3.7	operator+=	50
6.11.3.8	operator=	50
6.11.3.9	Pack	50
6.11.3.10	SerializeWith	50
6.11.4	Member Data Documentation	50
6.11.4.1	header	50
6.11.4.2	min_precinct	50
6.11.4.3	precincts	51
6.11.4.4	tile_header	51
6.12	jpeg2000::CodestreamIndex Class Reference	51
6.12.1	Detailed Description	52
6.12.2	Constructor & Destructor Documentation	52
6.12.2.1	CodestreamIndex	52
6.12.2.2	CodestreamIndex	52
6.12.2.3	~CodestreamIndex	52
6.12.3	Member Function Documentation	52
6.12.3.1	Clear	52
6.12.3.2	operator=	52
6.12.3.3	SerializeWith	52
6.12.4	Friends And Related Function Documentation	52
6.12.4.1	operator<<	52
6.12.5	Member Data Documentation	52
6.12.5.1	header	52
6.12.5.2	packets	52
6.12.5.3	PLT_markers	53
6.13	jpeg2000::CodingParameters Class Reference	53
6.13.1	Detailed Description	54
6.13.2	Member Typedef Documentation	54
6.13.2.1	Ptr	54
6.13.3	Member Enumeration Documentation	55
6.13.3.1	anonymous enum	55
6.13.4	Constructor & Destructor Documentation	55
6.13.4.1	CodingParameters	55
6.13.4.2	CodingParameters	55
6.13.4.3	~CodingParameters	55

6.13.5	Member Function Documentation	55
6.13.5.1	FillTotalPrecinctsVector	55
6.13.5.2	GetClosestResolution	55
6.13.5.3	GetPrecinctDataBinId	55
6.13.5.4	GetPrecincts	56
6.13.5.5	GetProgressionIndex	56
6.13.5.6	GetProgressionIndexLRCP	56
6.13.5.7	GetProgressionIndexRLCP	56
6.13.5.8	GetProgressionIndexRPCL	57
6.13.5.9	GetRoundDownResolution	58
6.13.5.10	GetRoundUpResolution	58
6.13.5.11	IsResolutionProgression	58
6.13.5.12	operator=	58
6.13.5.13	SerializeWith	58
6.13.6	Friends And Related Function Documentation	58
6.13.6.1	operator<<	58
6.13.7	Member Data Documentation	58
6.13.7.1	num_components	58
6.13.7.2	num_layers	59
6.13.7.3	num_levels	59
6.13.7.4	precinct_size	59
6.13.7.5	progression	59
6.13.7.6	size	59
6.13.7.7	total_precincts	59
6.14	AppInfo::Data Struct Reference	59
6.14.1	Detailed Description	60
6.14.2	Member Function Documentation	60
6.14.2.1	Reset	60
6.14.3	Member Data Documentation	60
6.14.3.1	child_iterations	60
6.14.3.2	child_pid	60
6.14.3.3	father_pid	60
6.14.3.4	num_connections	60
6.15	jpip::DataBinClass Class Reference	60
6.15.1	Detailed Description	61
6.15.2	Member Enumeration Documentation	61
6.15.2.1	anonymous enum	61
6.15.3	Constructor & Destructor Documentation	61
6.15.3.1	DataBinClass	61
6.15.4	Member Function Documentation	61

6.15.4.1	GetName	61
6.16	jpip::DataBinSelector< BIN_CLASS > Struct Template Reference	61
6.16.1	Detailed Description	61
6.17	jpip::DataBinSelector< DataBinClass::MAIN_HEADER > Struct Template Reference	62
6.17.1	Member Function Documentation	62
6.17.1.1	addTo	62
6.17.1.2	Get	62
6.18	jpip::DataBinSelector< DataBinClass::META_DATA > Struct Template Reference	62
6.18.1	Member Function Documentation	62
6.18.1.1	addTo	62
6.18.1.2	Get	63
6.19	jpip::DataBinSelector< DataBinClass::PRECINCT > Struct Template Reference	63
6.19.1	Member Function Documentation	63
6.19.1.1	addTo	63
6.19.1.2	Get	63
6.20	jpip::DataBinSelector< DataBinClass::TILE_HEADER > Struct Template Reference	63
6.20.1	Member Function Documentation	63
6.20.1.1	addTo	63
6.20.1.2	Get	64
6.21	jpip::DataBinServer Class Reference	64
6.21.1	Detailed Description	65
6.21.2	Member Enumeration Documentation	65
6.21.2.1	anonymous enum	65
6.21.3	Constructor & Destructor Documentation	65
6.21.3.1	DataBinServer	65
6.21.3.2	~DataBinServer	65
6.21.4	Member Function Documentation	65
6.21.4.1	end_woi	65
6.21.4.2	GenerateChunk	66
6.21.4.3	Reset	67
6.21.4.4	SetRequest	67
6.21.4.5	WritePlaceholder	67
6.21.4.6	WriteSegment	68
6.21.5	Member Data Documentation	68
6.21.5.1	cache_model	68
6.21.5.2	current_idx	68
6.21.5.3	data_writer	68
6.21.5.4	end_woi_	68
6.21.5.5	eof	69
6.21.5.6	file	69

6.21.5.7	files	69
6.21.5.8	has_woi	69
6.21.5.9	im_index	69
6.21.5.10	metareq	69
6.21.5.11	pending	69
6.21.5.12	range	69
6.21.5.13	woi	69
6.21.5.14	woi_composer	69
6.22	jpeg::DataBinWriter Class Reference	69
6.22.1	Detailed Description	71
6.22.2	Constructor & Destructor Documentation	71
6.22.2.1	DataBinWriter	71
6.22.2.2	~DataBinWriter	71
6.22.3	Member Function Documentation	71
6.22.3.1	ClearPreviousIds	71
6.22.3.2	GetCount	71
6.22.3.3	GetFree	71
6.22.3.4	operator bool	71
6.22.3.5	SetBuffer	72
6.22.3.6	SetCodestream	73
6.22.3.7	SetDataBinClass	73
6.22.3.8	Write	73
6.22.3.9	WriteEmpty	74
6.22.3.10	WriteEOR	75
6.22.3.11	WriteHeader	75
6.22.3.12	WritePlaceholder	75
6.22.3.13	WriteValue	76
6.22.3.14	WriteVBAS	76
6.22.4	Member Data Documentation	76
6.22.4.1	codestream_idx	76
6.22.4.2	databin_class	76
6.22.4.3	end	76
6.22.4.4	eof	76
6.22.4.5	ini	77
6.22.4.6	prev_codestream_idx	77
6.22.4.7	prev_databin_class	77
6.22.4.8	ptr	77
6.23	jpeg::EOR Class Reference	77
6.23.1	Detailed Description	77
6.23.2	Member Enumeration Documentation	77

6.23.2.1	anonymous enum	77
6.23.3	Constructor & Destructor Documentation	78
6.23.3.1	EOR	78
6.24	ipc::Event Class Reference	78
6.24.1	Detailed Description	79
6.24.2	Member Typedef Documentation	79
6.24.2.1	Ptr	79
6.24.3	Member Function Documentation	79
6.24.3.1	Dispose	79
6.24.3.2	Get	79
6.24.3.3	Init	80
6.24.3.4	Init	80
6.24.3.5	Pulse	80
6.24.3.6	Reset	80
6.24.3.7	Set	80
6.24.3.8	Wait	81
6.24.4	Member Data Documentation	81
6.24.4.1	condv	81
6.24.4.2	manual_reset	81
6.24.4.3	mutex	81
6.24.4.4	state	81
6.25	jpeg2000::FileManager Class Reference	81
6.25.1	Detailed Description	83
6.25.2	Constructor & Destructor Documentation	83
6.25.2.1	FileManager	83
6.25.2.2	FileManager	83
6.25.2.3	~FileManager	83
6.25.3	Member Function Documentation	83
6.25.3.1	cache_dir	83
6.25.3.2	ExistCacheImage	83
6.25.3.3	GetCacheFileName	83
6.25.3.4	Init	84
6.25.3.5	ReadBoxHeader	85
6.25.3.6	ReadCodestream	85
6.25.3.7	ReadCODMarker	85
6.25.3.8	ReadFlstBox	86
6.25.3.9	ReadImage	86
6.25.3.10	ReadJP2	86
6.25.3.11	ReadJPX	86
6.25.3.12	ReadNlstBox	87

6.25.3.13	ReadPLTMarker	87
6.25.3.14	ReadSIZMarker	87
6.25.3.15	ReadSODMarker	88
6.25.3.16	ReadSOTMarker	89
6.25.3.17	ReadUrlBox	89
6.25.3.18	root_dir	89
6.25.4	Member Data Documentation	89
6.25.4.1	cache_dir_	89
6.25.4.2	root_dir_	90
6.26	data::FileSegment Class Reference	90
6.26.1	Detailed Description	91
6.26.2	Constructor & Destructor Documentation	91
6.26.2.1	FileSegment	91
6.26.2.2	FileSegment	91
6.26.2.3	FileSegment	91
6.26.2.4	~FileSegment	91
6.26.3	Member Function Documentation	91
6.26.3.1	IsContiguousTo	91
6.26.3.2	operator!=	91
6.26.3.3	operator=	91
6.26.3.4	operator==	91
6.26.3.5	RemoveFirst	91
6.26.3.6	RemoveLast	92
6.26.3.7	SerializeWith	92
6.26.4	Friends And Related Function Documentation	92
6.26.4.1	operator<<	92
6.26.5	Member Data Documentation	92
6.26.5.1	length	92
6.26.5.2	Null	92
6.26.5.3	offset	92
6.27	http::Header Class Reference	92
6.27.1	Detailed Description	93
6.27.2	Member Typedef Documentation	93
6.27.2.1	CacheControl	93
6.27.2.2	ContentLength	93
6.27.2.3	ContentType	93
6.27.2.4	TransferEncoding	93
6.27.3	Constructor & Destructor Documentation	94
6.27.3.1	Header	94
6.27.3.2	Header	94

6.27.4 Friends And Related Function Documentation	94
6.27.4.1 operator==	94
6.28 http::HeaderBase< NAME > Class Template Reference	94
6.28.1 Detailed Description	95
6.28.2 Constructor & Destructor Documentation	95
6.28.2.1 HeaderBase	95
6.28.2.2 HeaderBase	95
6.28.3 Member Function Documentation	95
6.28.3.1 name	95
6.28.4 Friends And Related Function Documentation	95
6.28.4.1 operator<<	95
6.28.4.2 operator>>	95
6.28.5 Member Data Documentation	95
6.28.5.1 value	95
6.29 http::HeaderBase< HeaderName::UNDEFINED > Class Template Reference	96
6.29.1 Detailed Description	96
6.29.2 Constructor & Destructor Documentation	96
6.29.2.1 HeaderBase	96
6.29.2.2 HeaderBase	96
6.29.3 Friends And Related Function Documentation	97
6.29.3.1 operator<<	97
6.29.3.2 operator>>	97
6.29.4 Member Data Documentation	97
6.29.4.1 name	97
6.29.4.2 value	97
6.30 http::HeaderName Class Reference	97
6.30.1 Detailed Description	97
6.30.2 Member Data Documentation	98
6.30.2.1 CACHE_CONTROL	98
6.30.2.2 CONTENT_LENGTH	98
6.30.2.3 CONTENT_TYPE	98
6.30.2.4 TRANSFER_ENCODING	98
6.30.2.5 UNDEFINED	98
6.31 jpeg2000::ImageIndex Class Reference	98
6.31.1 Detailed Description	100
6.31.2 Member Typedef Documentation	100
6.31.2.1 Ptr	100
6.31.3 Constructor & Destructor Documentation	101
6.31.3.1 ImageIndex	101
6.31.3.2 ImageIndex	101

6.31.3.3	~ImageIndex	101
6.31.4	Member Function Documentation	101
6.31.4.1	BuildIndex	101
6.31.4.2	GetCodingParameters	101
6.31.4.3	GetHyperLink	101
6.31.4.4	GetMainHeader	101
6.31.4.5	GetMetadata	102
6.31.4.6	GetNumCodestreams	103
6.31.4.7	GetNumHyperLinks	103
6.31.4.8	GetNumMetadatas	103
6.31.4.9	GetOffsetPacket	103
6.31.4.10	GetPacket	103
6.31.4.11	GetPathName	103
6.31.4.12	GetPathName	103
6.31.4.13	GetPlaceholder	104
6.31.4.14	GetPLTLength	104
6.31.4.15	Init	104
6.31.4.16	Init	104
6.31.4.17	IsHyperLinked	105
6.31.4.18	operator CodingParameters::Ptr	105
6.31.4.19	operator=	105
6.31.4.20	ReadLock	105
6.31.4.21	ReadUnlock	105
6.31.5	Friends And Related Function Documentation	105
6.31.5.1	IndexManager	105
6.31.5.2	operator<<	105
6.31.6	Member Data Documentation	105
6.31.6.1	codestreams	105
6.31.6.2	coding_parameters	105
6.31.6.3	hyper_links	106
6.31.6.4	last_offset_packet	106
6.31.6.5	last_offset_PLT	106
6.31.6.6	last_packet	106
6.31.6.7	last_plt	106
6.31.6.8	max_resolution	106
6.31.6.9	meta_data	106
6.31.6.10	num_references	106
6.31.6.11	packet_indexes	106
6.31.6.12	path_name	106
6.31.6.13	rdwr_lock	106

6.32 jpeg2000::ImageInfo Class Reference	106
6.32.1 Detailed Description	107
6.32.2 Constructor & Destructor Documentation	107
6.32.2.1 ImageInfo	107
6.32.2.2 ImageInfo	107
6.32.2.3 ~ImageInfo	108
6.32.3 Member Function Documentation	108
6.32.3.1 operator=	108
6.32.3.2 SerializeWith	108
6.32.4 Friends And Related Function Documentation	108
6.32.4.1 operator<<	108
6.32.5 Member Data Documentation	108
6.32.5.1 codestreams	108
6.32.5.2 coding_parameters	108
6.32.5.3 meta_data	108
6.32.5.4 meta_data_hyperlinks	108
6.32.5.5 paths	108
6.33 jpeg2000::IndexManager Class Reference	108
6.33.1 Detailed Description	109
6.33.2 Constructor & Destructor Documentation	109
6.33.2.1 IndexManager	109
6.33.2.2 ~IndexManager	110
6.33.3 Member Function Documentation	110
6.33.3.1 CloseImage	110
6.33.3.2 file_manager	110
6.33.3.3 GetBegin	110
6.33.3.4 GetEnd	110
6.33.3.5 GetSize	110
6.33.3.6 Init	110
6.33.3.7 OpenImage	111
6.33.3.8 UnsafeCloseImage	112
6.33.3.9 UnsafeOpenImage	112
6.33.4 Member Data Documentation	112
6.33.4.1 file_manager_	112
6.33.4.2 index_list	112
6.33.4.3 mutex	112
6.34 net::InetAddress Class Reference	113
6.34.1 Detailed Description	113
6.34.2 Constructor & Destructor Documentation	113
6.34.2.1 InetAddress	113

6.34.2.2	InetAddress	114
6.34.2.3	InetAddress	114
6.34.2.4	InetAddress	114
6.34.3	Member Function Documentation	114
6.34.3.1	GetPath	114
6.34.3.2	GetPort	114
6.34.3.3	GetSize	114
6.34.3.4	GetSockAddr	114
6.34.3.5	operator=	114
6.34.4	Member Data Documentation	115
6.34.4.1	sock_addr	115
6.35	data::InputOperator Struct Reference	115
6.35.1	Detailed Description	115
6.35.2	Member Function Documentation	115
6.35.2.1	FileAccess	115
6.35.2.2	SerializeBytes	115
6.36	data::InputStream Class Reference	116
6.36.1	Detailed Description	116
6.36.2	Member Function Documentation	116
6.36.2.1	Serialize	116
6.37	ipc::IPCObject Class Reference	116
6.37.1	Detailed Description	117
6.37.2	Member Typedef Documentation	117
6.37.2.1	Ptr	117
6.37.3	Constructor & Destructor Documentation	117
6.37.3.1	IPCObject	117
6.37.3.2	~IPCObject	117
6.37.4	Member Function Documentation	118
6.37.4.1	Dispose	118
6.37.4.2	Init	118
6.37.4.3	IsValid	118
6.37.4.4	Wait	118
6.37.5	Member Data Documentation	118
6.37.5.1	valid	118
6.38	data::LockedAccess Struct Reference	119
6.38.1	Detailed Description	119
6.38.2	Member Function Documentation	119
6.38.2.1	configure	119
6.38.2.2	fgetc	119
6.38.2.3	fputc	119

6.38.2.4	fread	119
6.38.2.5	fwrite	119
6.39	jpeg2000::Metadata Class Reference	119
6.39.1	Detailed Description	120
6.39.2	Constructor & Destructor Documentation	120
6.39.2.1	Metadata	120
6.39.2.2	Metadata	120
6.39.2.3	~Metadata	120
6.39.3	Member Function Documentation	120
6.39.3.1	operator=	120
6.39.3.2	SerializeWith	120
6.39.4	Friends And Related Function Documentation	120
6.39.4.1	operator<<	120
6.39.5	Member Data Documentation	121
6.39.5.1	meta_data	121
6.39.5.2	place_holders	121
6.40	ipc::Mutex Class Reference	121
6.40.1	Detailed Description	122
6.40.2	Member Typedef Documentation	122
6.40.2.1	Ptr	122
6.40.3	Member Function Documentation	122
6.40.3.1	Dispose	122
6.40.3.2	Init	122
6.40.3.3	Init	122
6.40.3.4	Release	123
6.40.3.5	Wait	123
6.40.4	Member Data Documentation	123
6.40.4.1	locker	123
6.40.4.2	mutex	123
6.41	data::OutputOperator Struct Reference	123
6.41.1	Detailed Description	124
6.41.2	Member Function Documentation	124
6.41.2.1	FileAccess	124
6.41.2.2	SerializeBytes	124
6.42	data::OutputStream Class Reference	124
6.42.1	Detailed Description	124
6.42.2	Member Function Documentation	125
6.42.2.1	Serialize	125
6.43	jpeg2000::Packet Class Reference	125
6.43.1	Detailed Description	126

6.43.2	Constructor & Destructor Documentation	126
6.43.2.1	Packet	126
6.43.2.2	Packet	126
6.43.2.3	Packet	126
6.43.2.4	~Packet	126
6.43.3	Member Function Documentation	126
6.43.3.1	operator=	126
6.43.4	Friends And Related Function Documentation	126
6.43.4.1	operator<<	126
6.43.5	Member Data Documentation	126
6.43.5.1	component	126
6.43.5.2	layer	126
6.43.5.3	precinct_xy	126
6.43.5.4	resolution	126
6.44	jpeg2000::PacketIndex Class Reference	127
6.44.1	Detailed Description	127
6.44.2	Member Enumeration Documentation	128
6.44.2.1	anonymous enum	128
6.44.3	Constructor & Destructor Documentation	128
6.44.3.1	PacketIndex	128
6.44.3.2	PacketIndex	128
6.44.3.3	PacketIndex	128
6.44.3.4	~PacketIndex	128
6.44.4	Member Function Documentation	128
6.44.4.1	Add	128
6.44.4.2	Clear	128
6.44.4.3	operator=	128
6.44.4.4	operator[]	129
6.44.4.5	Size	130
6.44.5	Member Data Documentation	130
6.44.5.1	aux	130
6.44.5.2	offsets	130
6.45	jpeg::Request::ParametersMask Union Reference	130
6.45.1	Detailed Description	131
6.45.2	Constructor & Destructor Documentation	131
6.45.2.1	ParametersMask	131
6.45.3	Member Function Documentation	131
6.45.3.1	Clear	131
6.45.3.2	HasWOI	131
6.45.4	Member Data Documentation	131

6.45.4.1	cclose	131
6.45.4.2	cid	131
6.45.4.3	cnew	131
6.45.4.4	context	131
6.45.4.5	fsiz	131
6.45.4.6	items	131
6.45.4.7	len	132
6.45.4.8	metareq	132
6.45.4.9	model	132
6.45.4.10	roff	132
6.45.4.11	rsiz	132
6.45.4.12	stream	132
6.45.4.13	target	132
6.45.4.14	value	132
6.46	jpeg2000::Placeholder Class Reference	132
6.46.1	Detailed Description	133
6.46.2	Constructor & Destructor Documentation	133
6.46.2.1	Placeholder	133
6.46.2.2	Placeholder	133
6.46.2.3	Placeholder	133
6.46.2.4	~Placeholder	133
6.46.3	Member Function Documentation	133
6.46.3.1	length	133
6.46.3.2	operator=	133
6.46.3.3	SerializeWith	134
6.46.4	Friends And Related Function Documentation	134
6.46.4.1	operator<<	134
6.46.5	Member Data Documentation	134
6.46.5.1	data_length	134
6.46.5.2	header	134
6.46.5.3	id	134
6.46.5.4	is_jp2c	134
6.47	jpeg2000::Point Class Reference	134
6.47.1	Detailed Description	135
6.47.2	Constructor & Destructor Documentation	135
6.47.2.1	Point	135
6.47.2.2	Point	136
6.47.2.3	Point	137
6.47.2.4	~Point	137
6.47.3	Member Function Documentation	137

6.47.3.1	operator*=	137
6.47.3.2	operator++	137
6.47.3.3	operator+=	137
6.47.3.4	operator--	137
6.47.3.5	operator-=	137
6.47.3.6	operator/=	138
6.47.3.7	operator=	138
6.47.3.8	SerializeWith	138
6.47.4	Friends And Related Function Documentation	138
6.47.4.1	operator"!="	138
6.47.4.2	operator*	138
6.47.4.3	operator*	138
6.47.4.4	operator+	138
6.47.4.5	operator+	138
6.47.4.6	operator-	139
6.47.4.7	operator-	139
6.47.4.8	operator/	139
6.47.4.9	operator/	139
6.47.4.10	operator<<	139
6.47.4.11	operator==	139
6.47.5	Member Data Documentation	139
6.47.5.1	x	139
6.47.5.2	y	139
6.48	net::PollFD Struct Reference	139
6.48.1	Detailed Description	140
6.48.2	Constructor & Destructor Documentation	140
6.48.2.1	PollFD	140
6.48.3	Member Function Documentation	140
6.48.3.1	operator==	140
6.49	net::PollTable Class Reference	140
6.49.1	Detailed Description	141
6.49.2	Constructor & Destructor Documentation	141
6.49.2.1	PollTable	141
6.49.2.2	~PollTable	141
6.49.3	Member Function Documentation	141
6.49.3.1	Add	141
6.49.3.2	GetSize	141
6.49.3.3	operator[]	141
6.49.3.4	Poll	141
6.49.3.5	Remove	142

6.49.3.6	RemoveAt	142
6.49.4	Member Data Documentation	142
6.49.4.1	fds	142
6.50	http::Protocol Class Reference	142
6.50.1	Detailed Description	143
6.50.2	Constructor & Destructor Documentation	143
6.50.2.1	Protocol	143
6.50.2.2	Protocol	143
6.50.3	Member Function Documentation	143
6.50.3.1	mayorVersion	143
6.50.3.2	minorVersion	143
6.50.4	Friends And Related Function Documentation	144
6.50.4.1	operator<<	144
6.50.4.2	operator>>	144
6.50.5	Member Data Documentation	144
6.50.5.1	CRLF	144
6.50.5.2	mayorVersion_	144
6.50.5.3	minorVersion_	144
6.51	jpeg2000::Range Class Reference	144
6.51.1	Detailed Description	145
6.51.2	Constructor & Destructor Documentation	145
6.51.2.1	Range	145
6.51.2.2	Range	145
6.51.2.3	Range	145
6.51.2.4	~Range	145
6.51.3	Member Function Documentation	145
6.51.3.1	GetIndex	145
6.51.3.2	GetItem	146
6.51.3.3	IsValid	146
6.51.3.4	Length	146
6.51.3.5	operator=	146
6.51.4	Friends And Related Function Documentation	146
6.51.4.1	operator!=	146
6.51.4.2	operator<<	146
6.51.4.3	operator==	146
6.51.5	Member Data Documentation	146
6.51.5.1	first	146
6.51.5.2	last	146
6.52	ipc::RdWrLock Class Reference	147
6.52.1	Detailed Description	147

6.52.2	Member Typedef Documentation	147
6.52.2.1	Ptr	147
6.52.3	Member Function Documentation	147
6.52.3.1	Dispose	147
6.52.3.2	Init	148
6.52.3.3	Release	148
6.52.3.4	Wait	148
6.52.3.5	WaitForWriting	148
6.52.4	Member Data Documentation	148
6.52.4.1	rwlock	148
6.53	jpip::Request Class Reference	149
6.53.1	Detailed Description	150
6.53.2	Member Enumeration Documentation	150
6.53.2.1	RoundDirection	150
6.53.3	Constructor & Destructor Documentation	150
6.53.3.1	Request	150
6.53.3.2	~Request	150
6.53.4	Member Function Documentation	150
6.53.4.1	GetCodedChar	150
6.53.4.2	GetResolution	151
6.53.4.3	ParseModel	152
6.53.4.4	ParseParameter	152
6.53.4.5	ParseParameters	152
6.53.5	Member Data Documentation	152
6.53.5.1	cache_model	152
6.53.5.2	length_response	152
6.53.5.3	mask	153
6.53.5.4	max_codestream	153
6.53.5.5	min_codestream	153
6.53.5.6	resolution_size	153
6.53.5.7	round_direction	153
6.53.5.8	woi_position	153
6.53.5.9	woi_size	153
6.54	http::Request Class Reference	153
6.54.1	Detailed Description	154
6.54.2	Member Enumeration Documentation	154
6.54.2.1	Type	154
6.54.3	Constructor & Destructor Documentation	155
6.54.3.1	Request	155
6.54.4	Member Function Documentation	156

6.54.4.1	Parse	156
6.54.4.2	ParseParameter	156
6.54.4.3	ParseParameters	156
6.54.4.4	ParseURI	156
6.54.5	Friends And Related Function Documentation	157
6.54.5.1	operator<<	157
6.54.5.2	operator>>	157
6.54.6	Member Data Documentation	157
6.54.6.1	object	157
6.54.6.2	parameters	157
6.54.6.3	protocol	157
6.54.6.4	type	157
6.55	http::Response Class Reference	157
6.55.1	Detailed Description	158
6.55.2	Constructor & Destructor Documentation	158
6.55.2.1	Response	158
6.55.3	Friends And Related Function Documentation	158
6.55.3.1	operator<<	158
6.55.3.2	operator>>	158
6.55.4	Member Data Documentation	158
6.55.4.1	code	158
6.55.4.2	protocol	159
6.55.4.3	StatusCodes	159
6.55.4.4	statusCodesInitializer	159
6.56	data::Serializer< T > Struct Template Reference	159
6.56.1	Detailed Description	159
6.56.2	Member Function Documentation	159
6.56.2.1	Load	159
6.56.2.2	Save	160
6.57	data::Serializer< bool > Struct Template Reference	160
6.57.1	Detailed Description	160
6.57.2	Member Function Documentation	160
6.57.2.1	Load	160
6.57.2.2	Save	160
6.58	data::Serializer< int > Struct Template Reference	160
6.58.1	Detailed Description	161
6.58.2	Member Function Documentation	161
6.58.2.1	Load	161
6.58.2.2	Save	161
6.59	data::Serializer< multimap< string, int > > Struct Template Reference	161

6.59.1 Detailed Description	161
6.59.2 Member Function Documentation	162
6.59.2.1 Load	162
6.59.2.2 Save	162
6.60 data::Serializer< string > Struct Template Reference	162
6.60.1 Detailed Description	162
6.60.2 Member Function Documentation	162
6.60.2.1 Load	162
6.60.2.2 Save	162
6.61 data::Serializer< uint64_t > Struct Template Reference	163
6.61.1 Detailed Description	163
6.61.2 Member Function Documentation	163
6.61.2.1 Load	163
6.61.2.2 Save	163
6.62 data::Serializer< vector< T > > Struct Template Reference	163
6.62.1 Detailed Description	164
6.62.2 Member Function Documentation	164
6.62.2.1 Load	164
6.62.2.2 Save	164
6.63 net::Socket Class Reference	164
6.63.1 Detailed Description	165
6.63.2 Constructor & Destructor Documentation	165
6.63.2.1 Socket	165
6.63.2.2 Socket	166
6.63.2.3 Socket	166
6.63.2.4 ~Socket	166
6.63.3 Member Function Documentation	166
6.63.3.1 Accept	166
6.63.3.2 BindTo	166
6.63.3.3 Close	166
6.63.3.4 ConnectTo	166
6.63.3.5 IsBlockingMode	167
6.63.3.6 IsValid	167
6.63.3.7 IsValid	167
6.63.3.8 ListenAt	167
6.63.3.9 OpenInet	167
6.63.3.10 OpenUnix	168
6.63.3.11 operator int	169
6.63.3.12 operator=	169
6.63.3.13 Receive	169

6.63.3.14 ReceiveDescriptor	169
6.63.3.15 ReceiveFrom	169
6.63.3.16 Send	170
6.63.3.17 SendDescriptor	170
6.63.3.18 SendTo	170
6.63.3.19 SetBlockingMode	171
6.63.3.20 SetNoDelay	171
6.63.3.21 WaitForInput	171
6.63.3.22 WaitForOutput	171
6.63.4 Member Data Documentation	172
6.63.4.1 sid	172
6.64 net::SocketBuffer Class Reference	172
6.64.1 Detailed Description	173
6.64.2 Member Enumeration Documentation	173
6.64.2.1 anonymous enum	173
6.64.3 Constructor & Destructor Documentation	173
6.64.3.1 SocketBuffer	173
6.64.3.2 ~SocketBuffer	173
6.64.4 Member Function Documentation	173
6.64.4.1 GetReadBytes	173
6.64.4.2 GetSocket	173
6.64.4.3 overflow	173
6.64.4.4 sync	173
6.64.4.5 underflow	173
6.64.5 Member Data Documentation	173
6.64.5.1 in_buf	173
6.64.5.2 in_len	173
6.64.5.3 out_buf	173
6.64.5.4 out_len	174
6.64.5.5 socket	174
6.64.5.6 sum	174
6.65 net::SocketStream Class Reference	174
6.65.1 Detailed Description	174
6.65.2 Constructor & Destructor Documentation	174
6.65.2.1 SocketStream	174
6.65.2.2 ~SocketStream	174
6.65.3 Member Function Documentation	174
6.65.3.1 operator->	174
6.66 http::Response::StatusCodesInitializer Class Reference	175
6.66.1 Detailed Description	175

6.66.2	Constructor & Destructor Documentation	175
6.66.2.1	StatusCodesInitializer	175
6.67	TraceSystem Class Reference	175
6.67.1	Detailed Description	176
6.67.2	Constructor & Destructor Documentation	176
6.67.2.1	TraceSystem	176
6.67.2.2	~TraceSystem	176
6.67.3	Member Function Documentation	176
6.67.3.1	AppendToFile	176
6.67.3.2	AppendToFile	176
6.67.3.3	AppendToFile_	176
6.67.3.4	errorStream	176
6.67.3.5	logStream	176
6.67.3.6	traceStream	176
6.67.4	Member Data Documentation	176
6.67.4.1	appender	176
6.67.4.2	category	176
6.67.4.3	file_appender	176
6.67.4.4	file_layout	176
6.67.4.5	layout	176
6.67.4.6	traceSystem	176
6.68	ui Struct Reference	177
6.68.1	Member Function Documentation	177
6.68.1.1	read	177
6.69	net::UnixAddress Class Reference	177
6.69.1	Detailed Description	178
6.69.2	Constructor & Destructor Documentation	178
6.69.2.1	UnixAddress	178
6.69.2.2	UnixAddress	178
6.69.2.3	UnixAddress	178
6.69.3	Member Function Documentation	178
6.69.3.1	GetPath	178
6.69.3.2	GetSize	178
6.69.3.3	GetSockAddr	178
6.69.3.4	operator=	178
6.69.3.5	Reset	178
6.69.4	Member Data Documentation	179
6.69.4.1	sock_addr	179
6.70	data::UnlockedAccess Struct Reference	179
6.70.1	Detailed Description	179

6.70.2	Member Function Documentation	179
6.70.2.1	configure	179
6.70.2.2	fgetc	179
6.70.2.3	fputc	179
6.70.2.4	fread	179
6.70.2.5	fwrite	179
6.71	data::vint_vector Class Reference	180
6.71.1	Detailed Description	180
6.71.2	Constructor & Destructor Documentation	181
6.71.2.1	vint_vector	181
6.71.2.2	vint_vector	181
6.71.2.3	vint_vector	181
6.71.2.4	~vint_vector	181
6.71.3	Member Function Documentation	181
6.71.3.1	back	181
6.71.3.2	clear	181
6.71.3.3	data_bytes	181
6.71.3.4	num_bytes	181
6.71.3.5	operator=	181
6.71.3.6	operator[]	181
6.71.3.7	push_back	182
6.71.3.8	set_num_bytes	182
6.71.3.9	size	182
6.71.4	Member Data Documentation	182
6.71.4.1	data	182
6.71.4.2	mask	182
6.71.4.3	num_bytes_	182
6.72	jpeg::WOI Class Reference	182
6.72.1	Detailed Description	183
6.72.2	Constructor & Destructor Documentation	183
6.72.2.1	WOI	183
6.72.2.2	WOI	183
6.72.2.3	WOI	184
6.72.2.4	~WOI	184
6.72.3	Member Function Documentation	184
6.72.3.1	operator=	184
6.72.4	Friends And Related Function Documentation	184
6.72.4.1	operator!=	184
6.72.4.2	operator<<	184
6.72.4.3	operator==	184

6.72.5	Member Data Documentation	184
6.72.5.1	position	184
6.72.5.2	resolution	184
6.72.5.3	size	184
6.73	jpeg::WOIComposer Class Reference	185
6.73.1	Detailed Description	185
6.73.2	Constructor & Destructor Documentation	186
6.73.2.1	WOIComposer	186
6.73.2.2	WOIComposer	186
6.73.2.3	WOIComposer	186
6.73.2.4	~WOIComposer	186
6.73.3	Member Function Documentation	186
6.73.3.1	GetCurrentPacket	186
6.73.3.2	GetNextPacket	186
6.73.3.3	operator=	186
6.73.3.4	Reset	187
6.73.4	Member Data Documentation	187
6.73.4.1	coding_parameters	187
6.73.4.2	current_packet	187
6.73.4.3	max_precinct_xy	187
6.73.4.4	max_resolution	187
6.73.4.5	min_precinct_xy	187
6.73.4.6	more_packets	187
6.73.4.7	pxy1	187
6.73.4.8	pxy2	187
7	File Documentation	189
7.1	app_config.cc File Reference	189
7.2	app_config.h File Reference	189
7.3	app_info.cc File Reference	189
7.3.1	Macro Definition Documentation	190
7.3.1.1	LOCK_FILE	190
7.4	app_info.h File Reference	190
7.5	args_parser.cc File Reference	190
7.6	args_parser.h File Reference	190
7.7	base.cc File Reference	191
7.8	base.h File Reference	191
7.9	client_info.cc File Reference	191
7.10	client_info.h File Reference	191
7.11	client_manager.cc File Reference	191

7.12	client_manager.h File Reference	192
7.13	data/data.h File Reference	192
7.14	data/file.cc File Reference	192
7.15	data/file.h File Reference	192
7.16	data/file_segment.cc File Reference	193
7.17	data/file_segment.h File Reference	193
7.18	data/serialize.cc File Reference	193
7.19	data/serialize.h File Reference	194
7.20	data/vint_vector.cc File Reference	194
7.21	data/vint_vector.h File Reference	195
7.22	esa_jpip_server.cc File Reference	195
7.22.1	Macro Definition Documentation	196
7.22.1.1	CONFIG_FILE	196
7.22.1.2	POLLRDHUP	196
7.22.1.3	SERVER_APP_NAME	196
7.22.1.4	SERVER_NAME	196
7.22.2	Function Documentation	196
7.22.2.1	ChildProcess	196
7.22.2.2	ClientThread	196
7.22.2.3	main	196
7.22.2.4	ParseArguments	196
7.22.2.5	SIGCHLD_handler	196
7.22.3	Variable Documentation	196
7.22.3.1	app_info	196
7.22.3.2	base_id	196
7.22.3.3	cfg	196
7.22.3.4	child_address	197
7.22.3.5	child_lost	197
7.22.3.6	child_socket	197
7.22.3.7	father_address	197
7.22.3.8	index_manager	197
7.22.3.9	poll_table	197
7.23	http/header.cc File Reference	197
7.24	http/header.h File Reference	197
7.25	http/http.h File Reference	198
7.26	http/protocol.cc File Reference	198
7.27	http/protocol.h File Reference	198
7.28	http/request.cc File Reference	198
7.29	jpip/request.cc File Reference	199
7.30	http/request.h File Reference	199

7.31	jpip/request.h File Reference	199
7.32	http/response.cc File Reference	200
7.33	http/response.h File Reference	200
7.34	ipc/event.cc File Reference	200
7.35	ipc/event.h File Reference	201
7.36	ipc/ipc.h File Reference	201
7.37	ipc/ipc_object.cc File Reference	201
7.38	ipc/ipc_object.h File Reference	201
7.39	ipc/mutex.cc File Reference	202
7.40	ipc/mutex.h File Reference	202
7.41	ipc/rdwr_lock.cc File Reference	202
7.42	ipc/rdwr_lock.h File Reference	203
7.43	jpeg2000/codestream_index.cc File Reference	203
7.44	jpeg2000/codestream_index.h File Reference	203
7.45	jpeg2000/coding_parameters.cc File Reference	203
7.46	jpeg2000/coding_parameters.h File Reference	204
7.47	jpeg2000/file_manager.cc File Reference	204
7.47.1	Macro Definition Documentation	205
7.47.1.1	ASOC_BOX_ID	205
7.47.1.2	COD_MARKER	205
7.47.1.3	DBTL_BOX_ID	205
7.47.1.4	EOC_MARKER	205
7.47.1.5	FLST_BOX_ID	205
7.47.1.6	FTBL_BOX_ID	205
7.47.1.7	JP2C_BOX_ID	205
7.47.1.8	JPCH_BOX_ID	205
7.47.1.9	NLST_BOX_ID	205
7.47.1.10	PLT_MARKER	205
7.47.1.11	SIZ_MARKER	205
7.47.1.12	SOC_MARKER	205
7.47.1.13	SOD_MARKER	205
7.47.1.14	SOT_MARKER	205
7.47.1.15	URL__BOX_ID	205
7.47.1.16	XML__BOX_ID	205
7.48	jpeg2000/file_manager.h File Reference	205
7.49	jpeg2000/image_index.cc File Reference	206
7.50	jpeg2000/image_index.h File Reference	206
7.51	jpeg2000/image_info.cc File Reference	206
7.52	jpeg2000/image_info.h File Reference	207
7.53	jpeg2000/index_manager.cc File Reference	207

7.54 jpeg2000/index_manager.h File Reference	207
7.55 jpeg2000/jpeg2000.h File Reference	208
7.56 jpeg2000/meta_data.cc File Reference	208
7.57 jpeg2000/meta_data.h File Reference	208
7.58 jpeg2000/packet.cc File Reference	208
7.59 jpeg2000/packet.h File Reference	208
7.60 jpeg2000/packet_index.cc File Reference	209
7.61 jpeg2000/packet_index.h File Reference	209
7.62 jpeg2000/place_holder.cc File Reference	209
7.63 jpeg2000/place_holder.h File Reference	209
7.64 jpeg2000/point.cc File Reference	210
7.65 jpeg2000/point.h File Reference	210
7.66 jpeg2000/range.cc File Reference	210
7.67 jpeg2000/range.h File Reference	210
7.68 jpip/cache_model.cc File Reference	211
7.69 jpip/cache_model.h File Reference	211
7.70 jpip/databin_server.cc File Reference	211
7.71 jpip/databin_server.h File Reference	212
7.72 jpip/databin_writer.cc File Reference	212
7.73 jpip/databin_writer.h File Reference	212
7.74 jpip/jpip.cc File Reference	213
7.75 jpip/jpip.h File Reference	213
7.76 jpip/woi.cc File Reference	213
7.77 jpip/woi.h File Reference	213
7.78 jpip/woi_composer.cc File Reference	214
7.79 jpip/woi_composer.h File Reference	214
7.80 net/address.cc File Reference	214
7.81 net/address.h File Reference	214
7.82 net/net.h File Reference	215
7.83 net/poll_table.cc File Reference	215
7.84 net/poll_table.h File Reference	215
7.85 net/socket.cc File Reference	215
7.85.1 Macro Definition Documentation	216
7.85.1.1 POLLRDHUP	216
7.86 net/socket.h File Reference	216
7.87 net/socket_stream.cc File Reference	217
7.88 net/socket_stream.h File Reference	217
7.89 packet_information.cc File Reference	217
7.89.1 Function Documentation	217
7.89.1.1 main	217

7.90	tr1_compat.h File Reference	218
7.90.1	Macro Definition Documentation	218
7.90.1.1	SHARED_PTR	218
7.91	trace.cc File Reference	218
7.92	trace.h File Reference	218
7.92.1	Macro Definition Documentation	219
7.92.1.1	_BLUE	219
7.92.1.2	_GREEN	219
7.92.1.3	_RED	219
7.92.1.4	_RESET_COLOR	219
7.92.1.5	_SET_COLOR	219
7.92.1.6	_YELLOW	219
7.92.1.7	CERR	219
7.92.1.8	ERROR	219
7.92.1.9	LOG	219
7.92.1.10	LOG4CPP_FIX_ERROR_COLLISION	219
7.92.1.11	LOGC	219
7.92.1.12	TRACE	219
7.93	version.h File Reference	219
7.93.1	Macro Definition Documentation	219
7.93.1.1	VERSION	219

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

data	Contains a set of classes to easy the handling of data and files, as well as the serialization . .	11
http	Contains the definition of a set of classes for working easily with the protocol HTTP	12
ipc	Contains classes for working with the IPC mechanisms available in Linux using the <code>pthread</code> library	13
jpeg2000	Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard	14
jpip	Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard . .	15
net	Contains classes to easy the utilization of sockets, specially implemented for UNIX systems . .	16

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

net::Address	17
net::InetAddress	113
net::UnixAddress	177
AppConfig	18
AppInfo	22
ArgsParser	26
base	28
data::BaseFile< IO >	29
data::BaseFile< LockedAccess >	29
data::BaseStream< StreamClass, StreamOperator >	36
data::BaseStream< InputStream, InputOperator >	36
data::InputStream	116
data::BaseStream< OutputStream, OutputOperator >	36
data::OutputStream	124
jpip::CacheModel	39
ClientInfo	43
ClientManager	45
jpip::CacheModel::Codestream	48
jpeg2000::CodestreamIndex	51
jpeg2000::CodingParameters	53
AppInfo::Data	59
jpip::DataBinClass	60
jpip::DataBinSelector< BIN_CLASS >	61
jpip::DataBinSelector< DataBinClass::MAIN_HEADER >	62
jpip::DataBinSelector< DataBinClass::META_DATA >	62
jpip::DataBinSelector< DataBinClass::PRECINCT >	63
jpip::DataBinSelector< DataBinClass::TILE_HEADER >	63
jpip::DataBinServer	64
jpip::DataBinWriter	69
jpip::EOR	77
jpeg2000::FileManager	81
data::FileSegment	90
http::HeaderBase< NAME >	94
http::HeaderBase< HeaderName::UNDEFINED >	96
http::Header	92
http::HeaderName	97
jpeg2000::ImageIndex	98

jpeg2000::ImageInfo	106
jpeg2000::IndexManager	108
data::InputOperator	115
iostream	
net::SocketStream	174
ipc::IPCObject	116
ipc::Event	78
ipc::Mutex	121
ipc::RdWrLock	147
data::LockedAccess	119
jpeg2000::Metadata	119
data::OutputOperator	123
jpeg2000::Packet	125
jpeg2000::PacketIndex	127
jpip::Request::ParametersMask	130
jpeg2000::Placeholder	132
jpeg2000::Point	134
pollfd	
net::PollFD	139
net::PollTable	140
http::Protocol	142
jpeg2000::Range	144
http::Request	153
jpip::Request	149
http::Response	157
data::Serializer< T >	159
data::Serializer< bool >	160
data::Serializer< int >	160
data::Serializer< multimap< string, int > >	161
data::Serializer< string >	162
data::Serializer< uint64_t >	163
data::Serializer< vector< T > >	163
net::Socket	164
http::Response::StatusCodesInitializer	175
streambuf	
net::SocketBuffer	172
net::SocketStream	174
TraceSystem	175
ui	177
data::UnlockedAccess	179
data::vint_vector	180
jpip::WOI	182
jpip::WOIComposer	185

Chapter 3

Class Index

3.1 Class List

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

net::Address	Abstract base class to wrap the <code>sockaddr</code> derived structures	17
AppConfig	Contains the configuration parameters of the application	18
AppInfo	Contains the run-time information of the application	22
ArgsParser	Class that allows to parse and handle the application command line parameters	26
base	Contains a set of useful static methods used by the application	28
data::BaseFile< IO >	This is a wrapper class for the <code>FILE</code> functions that provides all the functionality to handle files safely	29
data::BaseStream< StreamClass, StreamOperator >	This template is used as the base for the input/output stream classes	36
jpip::CacheModel	The cache model of a JPIP client is handled using this class	39
ClientInfo	Contains information of a connected client	43
ClientManager	Handles a client connection with a dedicated thread	45
jpip::CacheModel::Codestream	Sub-class of the cache model class used to identify a codestream	48
jpeg2000::CodestreamIndex	Class used for indexing the information of a JPEG2000 codestream	51
jpeg2000::CodingParameters	Contains the coding parameters of a JPEG2000 image codestream	53
AppInfo::Data	Contains the data block that is maintained in shared memory	59
jpip::DataBinClass	Class that contains the definitions of all the data-bin classes defined for the JPIP protocol	60
jpip::DataBinSelector< BIN_CLASS >	Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes	61
jpip::DataBinSelector< DataBinClass::MAIN_HEADER >		62
jpip::DataBinSelector< DataBinClass::META_DATA >		62
jpip::DataBinSelector< DataBinClass::PRECINCT >		63
jpip::DataBinSelector< DataBinClass::TILE_HEADER >		63

jpip::DataBinServer	Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;	64
jpip::DataBinWriter	Class used to generate data-bin segments and write them into a memory buffer	69
jpip::EOR	Class that contains all the definitions of the EOF messages defined for the JPIP protocol . . .	77
ipc::Event	IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API	78
jpeg2000::FileManager	Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency	81
data::FileSegment	Identifies a data segment of a file	90
http::Header	Class used to handle a HTTP header	92
http::HeaderBase< NAME >	Template class used to identify a HTTP header	94
http::HeaderBase< HeaderName::UNDEFINED >	Specialization of the HeaderBase template class with the HeaderName::UNDEFINED value	96
http::HeaderName	Container for the strings associated to the most common HTTP headers, used for the specialization of the class HeaderBase	97
jpeg2000::ImageIndex	Contains the indexing information of a JPEG2000 image file that is managed by the index manager	98
jpeg2000::ImageInfo	Contains the indexing information of a JPEG2000 image	106
jpeg2000::IndexManager	Manages the indexing information of a repository for images	108
net::InetAddress	Class to identify and handle an Internet address	113
data::InputOperator	This struct identifies a basic input operator to be applied to a File object	115
data::InputStream	Specialization of the BaseStream for input serializations	116
ipc::IPCObject	Class base of all the IPC classes that has the basic operations (Init , Wait and Dispose) to be overloaded	116
data::LockedAccess	Struct for wrapping the basic FILE locked functions for reading and writing defined in stdio.h . . .	119
jpeg2000::Metadata	Contains the indexing information associated to the meta-data of a JPEG2000 image file . . .	119
ipc::Mutex	IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API	121
data::OutputOperator	This struct identifies a basic output operator to be applied to a File object	123
data::OutputStream	Specialization of the BaseStream for output serializations	124
jpeg2000::Packet	Contains the information of a packet	125
jpeg2000::PacketIndex	Class used for indexing the packets of a codestream image	127
jpip::Request::ParametersMask	Union used to control the presence of the different JPIP parameters in a request	130

jpeg2000::Placeholder	Contains the information of a place-holder	132
jpeg2000::Point	Represents a couple of integer values that can be used to identify a coordinate as well as a size	134
net::PollFD	Wrapper structure for the structure <code>pollfd</code> used by the kernel <code>poll</code> functions	139
net::PollTable	This class allows to perform polls easily over a vector of descriptors	140
http::Protocol	Class used to identify the HTTP protocol	142
jpeg2000::Range	Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range	144
ipc::RdWrLock	IPC object that offers the functionality of a read/write lock, implemented by means of the pthread <code>rwlock</code> API	147
jpip::Request	Class derived from the HTTP Request class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP	149
http::Request	Class used to identify a HTTP request (GET or POST)	153
http::Response	Class used to identify a HTTP response	157
data::Serializer< T >	This template class allows to define a "serializer"	159
data::Serializer< bool >	Serializer for the <code>bool</code> type	160
data::Serializer< int >	Serializer for the <code>int</code> type	160
data::Serializer< multimap< string, int > >	Serializer for the <code>multimap<string, int></code> class	161
data::Serializer< string >	Serializer for the <code>string</code> class	162
data::Serializer< uint64_t >	Serializer for the <code>uint64_t</code> type	163
data::Serializer< vector< T > >	Serializer for the <code>vector</code> class	163
net::Socket	This class has been designed to work with UNIX sockets in an easy and object oriented way	164
net::SocketBuffer	Class derived from the STL <code>std::streambuf</code> to allow streaming with sockets	172
net::SocketStream	Class derived from <code>std::iostream</code> and SocketBuffer that represents a socket stream	174
http::Response::StatusCodesInitializer	Class used for the initializer	175
TraceSystem	Wrapper used by the application to handle the log/trace messages by means of the <code>log4cpp</code> library	175
ui	177
net::UnixAddress	Class to identify and handle an UNIX address	177
data::UnlockedAccess	Struct for wrapping the basic <code>FILE</code> unlocked functions for reading and writing defined in <code>stdio_exts.h</code>	179
data::vint_vector	This class has been implemented with the same philosophy that the class STL <code>vector</code> , but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g	180

jpip::WOI	
Class that identifies a WOI (Window Of Interest)	182
jpip::WOIComposer	
By means of this class it is possible to find out the which packets of an image are associated to a WOI	185

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

app_config.cc	189
app_config.h	189
app_info.cc	189
app_info.h	190
args_parser.cc	190
args_parser.h	190
base.cc	191
base.h	191
client_info.cc	191
client_info.h	191
client_manager.cc	191
client_manager.h	192
esa_jpip_server.cc	195
packet_information.cc	217
tr1_compat.h	218
trace.cc	218
trace.h	218
version.h	219
data/data.h	192
data/file.cc	192
data/file.h	192
data/file_segment.cc	193
data/file_segment.h	193
data/serialize.cc	193
data/serialize.h	194
data/vint_vector.cc	194
data/vint_vector.h	195
http/header.cc	197
http/header.h	197
http/http.h	198
http/protocol.cc	198
http/protocol.h	198
http/request.cc	198
http/request.h	199
http/response.cc	200
http/response.h	200
ipc/event.cc	200
ipc/event.h	201

ipc/ipc.h	201
ipc/ipc_object.cc	201
ipc/ipc_object.h	201
ipc/mutex.cc	202
ipc/mutex.h	202
ipc/rdwr_lock.cc	202
ipc/rdwr_lock.h	203
jpeg2000/codestream_index.cc	203
jpeg2000/codestream_index.h	203
jpeg2000/coding_parameters.cc	203
jpeg2000/coding_parameters.h	204
jpeg2000/file_manager.cc	204
jpeg2000/file_manager.h	205
jpeg2000/image_index.cc	206
jpeg2000/image_index.h	206
jpeg2000/image_info.cc	206
jpeg2000/image_info.h	207
jpeg2000/index_manager.cc	207
jpeg2000/index_manager.h	207
jpeg2000/jpeg2000.h	208
jpeg2000/meta_data.cc	208
jpeg2000/meta_data.h	208
jpeg2000/packet.cc	208
jpeg2000/packet.h	208
jpeg2000/packet_index.cc	209
jpeg2000/packet_index.h	209
jpeg2000/place_holder.cc	209
jpeg2000/place_holder.h	209
jpeg2000/point.cc	210
jpeg2000/point.h	210
jpeg2000/range.cc	210
jpeg2000/range.h	210
jpeg2000/cache_model.cc	211
jpeg2000/cache_model.h	211
jpeg2000/databin_server.cc	211
jpeg2000/databin_server.h	212
jpeg2000/databin_writer.cc	212
jpeg2000/databin_writer.h	212
jpeg2000/jpip.cc	213
jpeg2000/jpip.h	213
jpeg2000/request.cc	199
jpeg2000/request.h	199
jpeg2000/woi.cc	213
jpeg2000/woi.h	213
jpeg2000/woi_composer.cc	214
jpeg2000/woi_composer.h	214
net/address.cc	214
net/address.h	214
net/net.h	215
net/poll_table.cc	215
net/poll_table.h	215
net/socket.cc	215
net/socket.h	216
net/socket_stream.cc	217
net/socket_stream.h	217

Chapter 5

Namespace Documentation

5.1 data Namespace Reference

Contains a set of classes to easy the handling of data and files, as well as the serialization.

Classes

- class [BaseFile](#)
This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.
- class [BaseStream](#)
This template is used as the base for the input/output stream classes.
- class [FileSegment](#)
Identifies a data segment of a file.
- struct [InputOperator](#)
This struct identifies a basic input operator to be applied to a `File` object.
- class [InputStream](#)
Specialization of the `BaseStream` for input serializations.
- struct [LockedAccess](#)
Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.
- struct [OutputOperator](#)
This struct identifies a basic output operator to be applied to a `File` object.
- class [OutputStream](#)
Specialization of the `BaseStream` for output serializations.
- struct [Serializer](#)
This template class allows to define a "serializer".
- struct [Serializer< bool >](#)
Serializer for the `bool` type.
- struct [Serializer< int >](#)
Serializer for the `int` type.
- struct [Serializer< multimap< string, int > >](#)
Serializer for the `multimap<string, int>` class.
- struct [Serializer< string >](#)
Serializer for the `string` class.
- struct [Serializer< uint64_t >](#)
Serializer for the `uint64_t` type.
- struct [Serializer< vector< T > >](#)
Serializer for the `vector` class.

- struct [UnlockedAccess](#)

Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_exts.h`.

- class [vint_vector](#)

This class has been implemented with the same philosophy that the class `STL vector`, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

Typedefs

- typedef [BaseFile](#)< [LockedAccess](#) > [File](#)

Specialization of the class [BaseFile](#) with locked access.

- typedef [BaseFile](#)< [UnlockedAccess](#) > [FastFile](#)

Specialization of the class [BaseFile](#) with unlocked access.

5.1.1 Detailed Description

Contains a set of classes to easy the handling of data and files, as well as the serialization.

5.1.2 Typedef Documentation

5.1.2.1 typedef [BaseFile](#)<[UnlockedAccess](#)> [data::FastFile](#)

Specialization of the class [BaseFile](#) with unlocked access.

See also

[BaseFile](#)
[UnlockedAccess](#)

5.1.2.2 typedef [BaseFile](#)<[LockedAccess](#)> [data::File](#)

Specialization of the class [BaseFile](#) with locked access.

See also

[BaseFile](#)
[LockedAccess](#)

5.2 http Namespace Reference

Contains the definition of a set of classes for working easily with the protocol HTTP.

Classes

- class [Header](#)

Class used to handle a HTTP header.

- class [HeaderBase](#)

Template class used to identify a HTTP header.

- class [HeaderBase](#)< [HeaderName::UNDEFINED](#) >

Specialization of the [HeaderBase](#) template class with the [HeaderName::UNDEFINED](#) value.

- class [HeaderName](#)

Container for the strings associated to the most common HTTP headers, used for the specialization of the class [HeaderBase](#).

- class [Protocol](#)

Class used to identify the HTTP protocol.

- class [Request](#)

Class used to identify a HTTP request (GET or POST).

- class [Response](#)

Class used to identify a HTTP response.

Functions

- istream & [operator>>](#) (istream &in, [Request](#) &request)
- ostream & [operator<<](#) (ostream &out, const [Request](#) &request)

5.2.1 Detailed Description

Contains the definition of a set of classes for working easily with the protocol HTTP.

5.2.2 Function Documentation

5.2.2.1 ostream& http::operator<< (ostream & out, const [Request](#) & request)

5.2.2.2 istream& http::operator>> (istream & in, [Request](#) & request)

Here is the call graph for this function:

5.3 ipc Namespace Reference

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

Classes

- class [Event](#)

IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API.

- class [IPCObject](#)

Class base of all the IPC classes that has the basic operations (*Init*, *Wait* and *Dispose*) to be overloaded.

- class [Mutex](#)

IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API.

- class [RdWrLock](#)

IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

Enumerations

- enum [WaitResult](#) { [WAIT_OBJECT](#) = 0, [WAIT_TIMEOUT](#), [WAIT_ERROR](#) }

Enumeration of the possible values returned when a wait operation is performed for an IPC object.

5.3.1 Detailed Description

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

These classes have been implemented to offer an object-oriented mechanism similar to the one offered by Windows, because of its simplicity and flexibility.

5.3.2 Enumeration Type Documentation

5.3.2.1 `enum ipc::WaitResult`

Enumeration of the possible values returned when a wait operation is performed for an IPC object.

Enumerator

`WAIT_OBJECT` Wait successful (object got)

`WAIT_TIMEOUT` Time out.

`WAIT_ERROR` Error.

5.4 `jpeg2000` Namespace Reference

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

Classes

- class [CodestreamIndex](#)
Class used for indexing the information of a JPEG2000 codestream.
- class [CodingParameters](#)
Contains the coding parameters of a JPEG2000 image codestream.
- class [FileManager](#)
Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.
- class [ImageIndex](#)
Contains the indexing information of a JPEG2000 image file that is managed by the index manager.
- class [ImageInfo](#)
Contains the indexing information of a JPEG2000 image.
- class [IndexManager](#)
Manages the indexing information of a repository fo images.
- class [Metadata](#)
Contains the indexing information associated to the meta-data of a JPEG2000 image file.
- class [Packet](#)
Contains the information of a packet.
- class [PacketIndex](#)
Class used for indexing the packets of a codestream image.
- class [Placeholder](#)
Contains the information of a place-holder.
- class [Point](#)
Represents a couple of integer values that can be used to identify a coordinate as well as a size.
- class [Range](#)
Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

Typedefs

- typedef [Point Size](#)

It is a synonymous of the class [Point](#).

5.4.1 Detailed Description

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

5.4.2 Typedef Documentation

5.4.2.1 typedef [Point](#) [jpeg2000::Size](#)

It is a synonymous of the class [Point](#).

See also

[Point](#)

5.5 jpip Namespace Reference

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

Classes

- class [CacheModel](#)
The cache model of a JPIP client is handled using this class.
- class [DataBinClass](#)
Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.
- struct [DataBinSelector](#)
Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.
- struct [DataBinSelector](#)< [DataBinClass::MAIN_HEADER](#) >
- struct [DataBinSelector](#)< [DataBinClass::META_DATA](#) >
- struct [DataBinSelector](#)< [DataBinClass::PRECINCT](#) >
- struct [DataBinSelector](#)< [DataBinClass::TILE_HEADER](#) >
- class [DataBinServer](#)
Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.
- class [DataBinWriter](#)
Class used to generate data-bin segments and write them into a memory buffer.
- class [EOR](#)
Class that contains all the definitions of the EOF messages defined for the JPIP protocol.
- class [Request](#)
Class derived from the HTTP [Request](#) class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.
- class [WOI](#)
Class that identifies a [WOI](#) (Window Of Interest).
- class [WOIComposer](#)
By means of this class it is possible to find out the which packets of an image are associated to a [WOI](#).

5.5.1 Detailed Description

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

5.6 net Namespace Reference

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

Classes

- class [Address](#)
Abstract base class to wrap the `sockaddr` derived structures.
- class [InetAddress](#)
Class to identify and handle an Internet address.
- struct [PollFD](#)
Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.
- class [PollTable](#)
This class allows to perform polls easily over a vector of descriptors.
- class [Socket](#)
This class has been designed to work with UNIX sockets in an easy and object oriented way.
- class [SocketBuffer](#)
Class derived from the STL `std::streambuf` to allow streaming with sockets.
- class [SocketStream](#)
Class derived from `std::iostream` and [SocketBuffer](#) that represents a socket stream.
- class [UnixAddress](#)
Class to identify and handle an UNIX address.

5.6.1 Detailed Description

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

Chapter 6

Class Documentation

6.1 net::Address Class Reference

Abstract base class to wrap the `sockaddr` derived structures.

```
#include <address.h>
```

Inheritance diagram for `net::Address`:

Collaboration diagram for `net::Address`:

Public Member Functions

- [Address \(\)](#)
Empty constructor.
- virtual `sockaddr *` [GetSockAddr \(\)](#) const =0
Returns a pointer to a `sockaddr` structure.
- virtual int [GetSize \(\)](#) const =0
Returns the size in bytes of the `sockaddr` structure returned by the previous method.
- virtual [~Address \(\)](#)
Empty destructor.

6.1.1 Detailed Description

Abstract base class to wrap the `sockaddr` derived structures.

This class is the base of the address classes.

See also

[InetAddress](#)
[UnixAddress](#)

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `net::Address::Address ()` [inline]

Empty constructor.

6.1.2.2 `virtual net::Address::~~Address () [inline],[virtual]`

Empty destructor.

6.1.3 Member Function Documentation

6.1.3.1 `virtual int net::Address::GetSize () const [pure virtual]`

Returns the size in bytes of the `sockaddr` structure returned by the previous method.

Implemented in [net::UnixAddress](#), and [net::InetAddress](#).

Here is the caller graph for this function:

6.1.3.2 `virtual sockaddr* net::Address::GetSockAddr () const [pure virtual]`

Returns a pointer to a `sockaddr` structure.

Implemented in [net::UnixAddress](#), and [net::InetAddress](#).

Here is the caller graph for this function:

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

- [net/address.h](#)

6.2 AppConfig Class Reference

Contains the configuration parameters of the application.

```
#include <app_config.h>
```

Collaboration diagram for AppConfig:

Public Member Functions

- [AppConfig](#) ()
Initializes the object with zero and empty values.
- bool [Load](#) (const char *file_name)
Loads the parameters from a configuration file.
- int [port](#) () const
Returns the listening port.
- string [address](#) () const
Returns the listening address.
- string [images_folder](#) () const
Returns the folder of the images.
- string [caching_folder](#) () const
Returns the folder used for caching.
- string [logging_folder](#) () const
Returns the folder used for the logging files.
- int [max_chunk_size](#) () const
Returns the maximum chunk size.
- int [max_connections](#) () const
Returns the maximum number of connections.
- bool [logging](#) () const

- *Returns true if the logging messages are allowed.*
- bool `log_requests` () const
Returns true if the client requests are logged.
- int `com_time_out` () const
Returns the connection time-out.
- int `cache_max_time` () const
Returns the maximum time for the cache files in seconds.
- virtual `~AppConfig` ()

Private Attributes

- int `port_`
Listening port.
- int `logging_`
true if logs messages are allowed
- int `log_requests_`
true if the client requests are logged
- string `address_`
Listening address.
- string `images_folder_`
Directory for the images.
- string `caching_folder_`
Directory for the caching files.
- string `logging_folder_`
Directory for the logging files.
- int `max_chunk_size_`
Maximum chunk size.
- int `max_connections_`
Maximum number of connections.
- int `com_time_out_`
Connection time-out.
- int `cache_max_time_`
Maximum time for the cache files.

Friends

- ostream & `operator<<` (ostream &out, const `AppConfig` &cfg)

6.2.1 Detailed Description

Contains the configuration parameters of the application.

It is possible to load these parameters from a configuration file. This class can be printed.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 `AppConfig::AppConfig ()` `[inline]`

Initializes the object with zero and empty values.

6.2.2.2 `virtual AppConfig::~AppConfig () [inline],[virtual]`

6.2.3 Member Function Documentation

6.2.3.1 `string AppConfig::address () const [inline]`

Returns the listening address.

Here is the caller graph for this function:

6.2.3.2 `int AppConfig::cache_max_time () const [inline]`

Returns the maximum time for the cache files in seconds.

Here is the caller graph for this function:

6.2.3.3 `string AppConfig::caching_folder () const [inline]`

Returns the folder used for caching.

Here is the caller graph for this function:

6.2.3.4 `int AppConfig::com_time_out () const [inline]`

Returns the connection time-out.

Here is the caller graph for this function:

6.2.3.5 `string AppConfig::images_folder () const [inline]`

Returns the folder of the images.

Here is the caller graph for this function:

6.2.3.6 `bool AppConfig::Load (const char * file_name)`

Loads the parameters from a configuration file.

Parameters

<i>file_name</i>	Configuration file.
------------------	---------------------

Returns

`true` if successful.

Here is the caller graph for this function:

6.2.3.7 `bool AppConfig::log_requests () const [inline]`

Returns `true` if the client requests are logged.

Here is the caller graph for this function:

6.2.3.8 `bool AppConfig::logging () const [inline]`

Returns `true` if the logging messages are allowed.

Here is the caller graph for this function:

6.2.3.9 `string AppConfig::logging_folder () const [inline]`

Returns the folder used for the logging files.

Here is the caller graph for this function:

6.2.3.10 `int AppConfig::max_chunk_size () const [inline]`

Returns the maximum chunk size.

Here is the caller graph for this function:

6.2.3.11 `int AppConfig::max_connections () const [inline]`

Returns the maximum number of connections.

Here is the caller graph for this function:

6.2.3.12 `int AppConfig::port () const [inline]`

Returns the listening port.

Here is the caller graph for this function:

6.2.4 Friends And Related Function Documentation

6.2.4.1 `ostream& operator<< (ostream & out, const AppConfig & cfg) [friend]`

6.2.5 Member Data Documentation

6.2.5.1 `string AppConfig::address_ [private]`

Listening address.

6.2.5.2 `int AppConfig::cache_max_time_ [private]`

Maximum time for the cache files.

6.2.5.3 `string AppConfig::caching_folder_ [private]`

Directory for the caching files.

6.2.5.4 `int AppConfig::com_time_out_ [private]`

Connection time-out.

6.2.5.5 string AppConfig::images_folder_ [private]

Directory for the images.

6.2.5.6 int AppConfig::log_requests_ [private]

true if the client requests are logged

6.2.5.7 int AppConfig::logging_ [private]

true if logs messages are allowed

6.2.5.8 string AppConfig::logging_folder_ [private]

Directory for the logging files.

6.2.5.9 int AppConfig::max_chunk_size_ [private]

Maximum chunk size.

6.2.5.10 int AppConfig::max_connections_ [private]

Maximum number of connections.

6.2.5.11 int AppConfig::port_ [private]

Listening port.

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

- [app_config.h](#)
- [app_config.cc](#)

6.3 ApplInfo Class Reference

Contains the run-time information of the application.

```
#include <app_info.h>
```

Collaboration diagram for ApplInfo:

Classes

- struct [Data](#)
Contains the data block that is maintained in shared memory.

Public Member Functions

- [ApplInfo](#) ()
Initializes the object.
- bool [Init](#) ()

- Initializes the object and the handling of the application run-time information.*
- `bool is_running () const`
Returns true if the application is running.
- `ApplInfo & Update ()`
Updates the run-time information of the application.
- `double available_memory () const`
Returns the available memory of the system.
- `double father_memory () const`
Returns the memory used by the father process.
- `double child_memory () const`
Returns the memory used by the child process.
- `int num_threads () const`
Returns the number of active threads.
- `unsigned long child_time () const`
Returns the time spent by the child process.
- `unsigned long time () const`
Returns the time spent by the father process.
- `Data * operator-> () const`
- `~ApplInfo ()`

Private Member Functions

- `string GetProcStat_ (int pid, int field) const`
Returns a specific field of /proc/<pid>/stat as a string.
- `template<typename TYPE >`
`TYPE GetProcStat (int pid, int field) const`
Returns a specific field of /proc/<pid>/stat as a defined type.

Private Attributes

- `int shmid`
Identifier of the shared memory block.
- `int lock_file`
Lock file.
- `Data * data_ptr`
Pointer to the shared memory block.
- `bool is_running_`
true if the application is running
- `int num_threads_`
Number of active threads.
- `double child_memory_`
Memory used by the child process.
- `unsigned long time_`
Time spent by the father.
- `double father_memory_`
Memory used by the father process.
- `double available_memory_`
Available memory in the system.
- `unsigned long child_time_`
Time spend by the child.

Friends

- ostream & [operator<<](#) (ostream &out, const [AppInfo](#) &app)

6.3.1 Detailed Description

Contains the run-time information of the application.

This class can be printed.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 AppInfo::AppInfo () [inline]

Initializes the object.

6.3.2.2 AppInfo::~AppInfo ()

6.3.3 Member Function Documentation

6.3.3.1 double AppInfo::available_memory () const [inline]

Returns the available memory of the system.

Here is the caller graph for this function:

6.3.3.2 double AppInfo::child_memory () const [inline]

Returns the memory used by the child process.

Here is the caller graph for this function:

6.3.3.3 unsigned long AppInfo::child_time () const [inline]

Returns the time spent by the child process.

Here is the caller graph for this function:

6.3.3.4 double AppInfo::father_memory () const [inline]

Returns the memory used by the father process.

Here is the caller graph for this function:

6.3.3.5 template<typename TYPE> TYPE AppInfo::GetProcStat (int *pid*, int *field*) const [inline], [private]

Returns a specific field of /proc/<pid>/stat as a defined type.

6.3.3.6 string AppInfo::GetProcStat_ (int *pid*, int *field*) const [private]

Returns a specific field of /proc/<pid>/stat as a string.

6.3.3.7 `bool ApplInfo::Init ()`

Initializes the object and the handling of the application run-time information.

Returns

`true` if successful.

Here is the caller graph for this function:

6.3.3.8 `bool ApplInfo::is_running () const [inline]`

Returns `true` if the application is running.

Here is the caller graph for this function:

6.3.3.9 `int ApplInfo::num_threads () const [inline]`

Returns the number of active threads.

Here is the caller graph for this function:

6.3.3.10 `Data* ApplInfo::operator-> () const [inline]`

6.3.3.11 `unsigned long ApplInfo::time () const [inline]`

Returns the time spent by the father process.

6.3.3.12 `ApplInfo & ApplInfo::Update ()`

Updates the run-time information of the application.

Here is the caller graph for this function:

6.3.4 Friends And Related Function Documentation

6.3.4.1 `ostream& operator<< (ostream & out, const ApplInfo & app) [friend]`

6.3.5 Member Data Documentation

6.3.5.1 `double ApplInfo::available_memory_ [private]`

Available memory in the system.

6.3.5.2 `double ApplInfo::child_memory_ [private]`

Memory used by the child process.

6.3.5.3 `unsigned long ApplInfo::child_time_ [private]`

Time spend by the child.

6.3.5.4 `Data* ApplInfo::data_ptr` [private]

Pointer to the shared memory block.

6.3.5.5 `double ApplInfo::father_memory_` [private]

Memory used by the father process.

6.3.5.6 `bool ApplInfo::is_running_` [private]

true if the application is running

6.3.5.7 `int ApplInfo::lock_file` [private]

Lock file.

6.3.5.8 `int ApplInfo::num_threads_` [private]

Number of active threads.

6.3.5.9 `int ApplInfo::shmid` [private]

Identifier of the shared memory block.

6.3.5.10 `unsigned long ApplInfo::time_` [private]

Time spent by the father.

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

- [app_info.h](#)
- [app_info.cc](#)

6.4 ArgsParser Class Reference

Class that allows to parse and handle the application command line parameters.

```
#include <args_parser.h>
```

Collaboration diagram for ArgsParser:

Public Member Functions

- [ArgsParser](#) ([AppConfig](#) &_cfg, [ApplInfo](#) &_app_info)
Initializes the object.
- `bool` [Parse](#) (int argc, char **argv)
Parses and handles the application command line parameters.

Private Attributes

- [AppConfig](#) & `cfg`
Application configuration.
- [AppInfo](#) & `app_info`
Application run-time information.

6.4.1 Detailed Description

Class that allows to parse and handle the application command line parameters.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ArgsParser::ArgsParser (AppConfig & _cfg, AppInfo & _app_info) `[inline]`

Initializes the object.

Parameters

<code>_cfg</code>	Application configuration.
<code>_app_info</code>	Application run-time information.

6.4.3 Member Function Documentation

6.4.3.1 bool ArgsParser::Parse (int argc, char ** argv)

Parses and handles the application command line parameters.

Parameters

<code>argc</code>	Number of parameters.
<code>argv</code>	Command line parameters.

Returns

`true` if successful.

Here is the call graph for this function:

6.4.4 Member Data Documentation

6.4.4.1 AppInfo& ArgsParser::app_info `[private]`

Application run-time information.

6.4.4.2 AppConfig& ArgsParser::cfg `[private]`

Application configuration.

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

- [args_parser.h](#)
- [args_parser.cc](#)

6.5 base Struct Reference

Contains a set of useful static methods used by the application.

```
#include <base.h>
```

Collaboration diagram for base:

Static Public Member Functions

- `template<typename TYPE >`
`static std::string to_string (TYPE val)`
Converts a value to a string.
- `template<typename T >`
`static void copy (std::vector< T > &dest, const std::vector< T > &src)`
Copies a vector.
- `template<typename T >`
`static void copy (std::vector< std::vector< T > > &dest, const std::vector< std::vector< T > > &src)`
Copies a vector of vectors.
- `template<typename T1 , typename T2 >`
`static void copy (std::multimap< T1, T2 > &dest, const std::multimap< T1, T2 > &src)`
Copies a multimap.

6.5.1 Detailed Description

Contains a set of useful static methods used by the application.

6.5.2 Member Function Documentation

6.5.2.1 `template<typename T > static void base::copy (std::vector< T > & dest, const std::vector< T > & src)`
`[inline], [static]`

Copies a vector.

Here is the caller graph for this function:

6.5.2.2 `template<typename T > static void base::copy (std::vector< std::vector< T > > & dest, const std::vector< std::vector< T > > & src)` `[inline], [static]`

Copies a vector of vectors.

Here is the call graph for this function:

6.5.2.3 `template<typename T1 , typename T2 > static void base::copy (std::multimap< T1, T2 > & dest, const std::multimap< T1, T2 > & src)` `[inline], [static]`

Copies a multimap.

6.5.2.4 `template<typename TYPE > static std::string base::to_string (TYPE val)` `[inline], [static]`

Converts a value to a string.

Parameters

<i>val</i>	Value to convert.
------------	-------------------

Here is the caller graph for this function:

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

- [base.h](#)

6.6 data::BaseFile< IO > Class Template Reference

This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.

```
#include <file.h>
```

Collaboration diagram for data::BaseFile< IO >:

Public Types

- typedef [SHARED_PTR< BaseFile< IO > > Ptr](#)
Safe pointer to this class.

Public Member Functions

- [BaseFile](#) ()
Initialized the internal file pointer to `NULL`.
- bool [Open](#) (const char *file_name, const char *access)
Opens a file with a specific access mode.
- bool [Open](#) (const string &file_name, const char *access)
Opens a file with a specific access mode.
- template<class IO2 >
bool [Open](#) (const [BaseFile](#)< IO2 > &file, const char *access)
Opens a file with a specific access mode given an already opened `File` object.
- bool [OpenForReading](#) (const char *file_name)
- bool [OpenForReading](#) (const string &file_name)
- template<class IO2 >
bool [OpenForReading](#) (const [BaseFile](#)< IO2 > &file)
- bool [OpenForWriting](#) (const char *file_name)
- bool [OpenForWriting](#) (const string &file_name)
- template<class IO2 >
bool [OpenForWriting](#) (const [BaseFile](#)< IO2 > &file)
- bool [Seek](#) (int offset, int origin=SEEK_SET) const
Changes the current position of the file.
- void [Close](#) ()
Closes the file.
- uint64_t [GetOffset](#) () const
Returns the current file position.
- int [IsEOF](#) () const
Returns the EOF status (`feof`) of the file.
- int [GetDescriptor](#) () const
Returns the file descriptor.
- uint64_t [GetSize](#) () const
Return the current size of the file, without modifying the file position.

- int [ReadByte](#) () const
Reads a byte from the file.
- template<typename T >
bool [Read](#) (T *value, int num_bytes=sizeof(T)) const
Reads a value from the file.
- template<typename T >
bool [ReadReverse](#) (T *value, int num_bytes=sizeof(T)) const
Reads a value from the file in reverse order.
- int [WriteByte](#) (int c) const
Writes a byte to the file.
- template<typename T >
bool [Write](#) (T *value, int num_bytes=sizeof(T)) const
Writes a value to the file.
- template<typename T >
bool [WriteReverse](#) (T *value, int num_bytes=sizeof(T)) const
Writes a value to the file in reverse order.
- bool [IsValid](#) () const
Returns `true` if the file pointer is not `NULL`.
- operator bool () const
Returns `true` if the file pointer is not `NULL`.
- virtual [~BaseFile](#) ()
The destructor closes the file.

Static Public Member Functions

- static bool [Exists](#) (const char *file_name)
Returns `true` if the given file exists.

Private Attributes

- FILE * [file_ptr](#)
File pointer.

6.6.1 Detailed Description

```
template<class IO>class data::BaseFile< IO >
```

This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.

It is defined as a template in order to allow to use either the locked or the unlocked API, by means of the structs [LockedAccess](#) and [UnlockedAccess](#). The unlocked API is not thread-safe, but it provides faster file operations.

See also

[LockedAccess](#)
[UnlockedAccess](#)

6.6.2 Member Typedef Documentation

6.6.2.1 template<class IO> typedef `SHARED_PTR< BaseFile<IO> > data::BaseFile< IO >::Ptr`

Safe pointer to this class.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 `template<class IO> data::BaseFile< IO >::BaseFile () [inline]`

Initialized the internal file pointer to `NULL`.

6.6.3.2 `template<class IO> virtual data::BaseFile< IO >::~~BaseFile () [inline],[virtual]`

The destructor closes the file.

6.6.4 Member Function Documentation

6.6.4.1 `template<class IO> void data::BaseFile< IO >::Close () [inline]`

Closes the file.

Here is the caller graph for this function:

6.6.4.2 `template<class IO> static bool data::BaseFile< IO >::Exists (const char * file_name) [inline],[static]`

Returns `true` if the given file exists.

This is a wrapper of the system function `stat`.

6.6.4.3 `template<class IO> int data::BaseFile< IO >::GetDescriptor () const [inline]`

Returns the file descriptor.

Here is the caller graph for this function:

6.6.4.4 `template<class IO> uint64_t data::BaseFile< IO >::GetOffset () const [inline]`

Returns the current file position.

Here is the caller graph for this function:

6.6.4.5 `template<class IO> uint64_t data::BaseFile< IO >::GetSize () const [inline]`

Return the current size of the file, without modifying the file position.

Here is the caller graph for this function:

6.6.4.6 `template<class IO> int data::BaseFile< IO >::IsEOF () const [inline]`

Returns the EOF status (`feof`) of the file.

6.6.4.7 `template<class IO> bool data::BaseFile< IO >::IsValid () const [inline]`

Returns `true` if the file pointer is not `NULL`.

Here is the caller graph for this function:

6.6.4.8 `template<class IO> bool data::BaseFile< IO >::Open (const char * file_name, const char * access)`
`[inline]`

Opens a file with a specific access mode.

Parameters

<i>file_name</i>	Path name of the file to open.
<i>access</i>	Access mode as a <code>fopen</code> compatible format string.

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.9 `template<class IO> bool data::BaseFile< IO >::Open (const string & file_name, const char * access)`
`[inline]`

Opens a file with a specific access mode.

Parameters

<i>file_name</i>	Path name of the file to open.
<i>access</i>	Access mode as a <code>fopen</code> compatible format string.

Returns

`true` if successful.

6.6.4.10 `template<class IO> template<class IO2 > bool data::BaseFile< IO >::Open (const BaseFile< IO2 > & file,`
`const char * access) [inline]`

Opens a file with a specific access mode given an already opened `File` object.

The descriptor of the opened file is duplicated and re-opened with the access mode given.

Parameters

<i>file</i>	Opened file.
<i>access</i>	Access mode as a <code>fopen</code> compatible format string.

Returns

`true` if successful.

6.6.4.11 `template<class IO> bool data::BaseFile< IO >::OpenForReading (const char * file_name) [inline]`

Here is the caller graph for this function:

6.6.4.12 `template<class IO> bool data::BaseFile< IO >::OpenForReading (const string & file_name) [inline]`

6.6.4.13 `template<class IO> template<class IO2 > bool data::BaseFile< IO >::OpenForReading (const BaseFile< IO2`
`> & file) [inline]`

6.6.4.14 `template<class IO> bool data::BaseFile< IO >::OpenForWriting (const char * file_name) [inline]`

6.6.4.15 `template<class IO> bool data::BaseFile< IO >::OpenForWriting (const string & file_name) [inline]`

6.6.4.16 `template<class IO> template<class IO2 > bool data::BaseFile< IO >::OpenForWriting (const BaseFile< IO2`
`> & file) [inline]`

6.6.4.17 `template<class IO> data::BaseFile< IO >::operator bool () const [inline]`

Returns `true` if the file pointer is not `NULL`.

6.6.4.18 `template<class IO> template<typename T> bool data::BaseFile< IO >::Read (T * value, int num_bytes = sizeof(T)) const [inline]`

Reads a value from the file.

Parameters

<i>value</i>	Pointer to the value where to store.
<i>num_bytes</i>	Number of bytes to read (by default, the size of the value).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.19 `template<class IO> int data::BaseFile< IO >::ReadByte () const [inline]`

Reads a byte from the file.

6.6.4.20 `template<class IO> template<typename T> bool data::BaseFile< IO >::ReadReverse (T * value, int num_bytes = sizeof(T)) const [inline]`

Reads a value from the file in reverse order.

Parameters

<i>value</i>	Pointer to the value where to store.
<i>num_bytes</i>	Number of bytes to read (by default, the size of the value).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.21 `template<class IO> bool data::BaseFile< IO >::Seek (int offset, int origin = SEEK_SET) const [inline]`

Changes the current position of the file.

Parameters

<i>offset</i>	Offset to add to the current position.
<i>origin</i>	Origin to use for the change (<code>SEEK_SET</code> by default).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.22 `template<class IO> template<typename T> bool data::BaseFile< IO >::Write (T * value, int num_bytes = sizeof(T)) const [inline]`

Writes a value to the file.

Parameters

<i>value</i>	Pointer to the value.
<i>num_bytes</i>	Number of bytes to write (by default, the size of the value).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.23 `template<class IO> int data::BaseFile< IO >::WriteByte (int c) const [inline]`

Writes a byte to the file.

6.6.4.24 `template<class IO> template<typename T > bool data::BaseFile< IO >::WriteReverse (T * value, int num_bytes = sizeof(T)) const [inline]`

Writes a value to the file in reverse order.

Parameters

<i>value</i>	Pointer to the value.
<i>num_bytes</i>	Number of bytes to write (by default, the size of the value).

Returns

`true` if successful.

6.6.5 Member Data Documentation

6.6.5.1 `template<class IO> FILE* data::BaseFile< IO >::file_ptr [private]`

File pointer.

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

- [data/file.h](#)

6.7 data::BaseStream< StreamClass, StreamOperator > Class Template Reference

This template is used as the base for the input/output stream classes.

```
#include <serialize.h>
```

Collaboration diagram for `data::BaseStream< StreamClass, StreamOperator >`:

Public Member Functions

- [BaseStream](#) ()
Initializes the status to false.
- `StreamClass & Open (const char *file_name)`
Opens a file for serialization.
- `StreamClass & Open (const char *file_name, const char *access)`
Opens a file for serialization.

- StreamClass & [Close](#) ()
Closes the file of the serialization and finish the serialization.
- StreamClass & [SerializeBytes](#) (void *ptr, int num_bytes)
Serializes a number of bytes.
- template<typename T >
StreamClass & [operator&](#) (T &var)
This operator overloading is the key of the serialization mechanism.
- bool [result](#) () const
Returns the internal serialization status.
- [operator bool](#) () const
Return the internal serialization status.
- virtual [~BaseStream](#) ()
The destructor automatically closes the file-.

Protected Attributes

- [File file_](#)
File used for the serialization.
- bool [result_](#)
Internal current status of the serialization.

6.7.1 Detailed Description

template<typename StreamClass, typename StreamOperator> class data::BaseStream< StreamClass, StreamOperator >

This template is used as the base for the input/output stream classes.

Contains the basic functionality for the serialization with files, composed by a file object and an internal status. This status is updated with each operation and, in a sequence of serialization, this is stopped just after this status is set to `false`.

In order to have a type serializable, it must comply with one of these requirements: i) to have implemented a "serializer" with the class [Serializer](#), or ii) to have defined a member method called `SerializeWith`.

The first option is useful for the basic types (int, float, etc.) and for those classes already defined and that can not be modified. The second option is more elegant for those classes that can be modified specifically for serialization.

The `SerializeWith` method must be defined as follows:

```
template<typename T> T& SerializeWith(T& stream) { return (stream & member1
& member2 & ...); }
```

See also

[Serializer](#)
[InputStream](#)
[OutputStream](#)

6.7.2 Constructor & Destructor Documentation

6.7.2.1 template<typename StreamClass, typename StreamOperator> data::BaseStream< StreamClass, StreamOperator >::BaseStream () [inline]

Initializes the status to `false`.

6.7.2.2 `template<typename StreamClass, typename StreamOperator> virtual data::BaseStream< StreamClass, StreamOperator >::~BaseStream () [inline], [virtual]`

The destructor automatically closes the file-.

6.7.3 Member Function Documentation

6.7.3.1 `template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::Close () [inline]`

Closes the file of the serialization and finish the serialization.

6.7.3.2 `template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::Open (const char * file_name) [inline]`

Opens a file for serialization.

Parameters

<i>file_name</i>	Path name of the file to open.
------------------	--------------------------------

Returns

`*this.`

Here is the caller graph for this function:

6.7.3.3 `template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::Open (const char * file_name, const char * access) [inline]`

Opens a file for serialization.

Parameters

<i>file_name</i>	Path name of the file to open.
<i>access</i>	Access mode to use to open the file.

Returns

`*this.`

6.7.3.4 `template<typename StreamClass, typename StreamOperator> data::BaseStream< StreamClass, StreamOperator >::operator bool () const [inline]`

Return the internal serialization status.

6.7.3.5 `template<typename StreamClass, typename StreamOperator> template<typename T > StreamClass& data::BaseStream< StreamClass, StreamOperator >::operator& (T & var) [inline]`

This operator overloading is the key of the serialization mechanism.

6.7.3.6 `template<typename StreamClass, typename StreamOperator> bool data::BaseStream< StreamClass, StreamOperator >::result () const [inline]`

Returns the internal serialization status.

6.7.3.7 `template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::SerializeBytes (void * ptr, int num_bytes) [inline]`

Serializes a number of bytes.

Depending on the stream operator in the template, this serialization is either a read or a write operation.

Parameters

<i>ptr</i>	Pointer to the buffer.
<i>num_bytes</i>	Number of bytes.

Returns

`*this.`

Here is the caller graph for this function:

6.7.4 Member Data Documentation

6.7.4.1 `template<typename StreamClass, typename StreamOperator> File data::BaseStream< StreamClass, StreamOperator >::file_ [protected]`

File used for the serialization.

6.7.4.2 `template<typename StreamClass, typename StreamOperator> bool data::BaseStream< StreamClass, StreamOperator >::result_ [protected]`

Internal current status of the serialization.

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

- [data/serialize.h](#)

6.8 jpip::CacheModel Class Reference

The cache model of a JPIP client is handled using this class.

```
#include <cache_model.h>
```

Collaboration diagram for jpip::CacheModel:

Classes

- class [Codestream](#)
Sub-class of the cache model class used to identify a codestream.

Public Member Functions

- [CacheModel](#) ()
Empty constructor.
- [CacheModel](#) (const [CacheModel](#) &model)
Copy constructor.
- [CacheModel](#) & [operator=](#) (const [CacheModel](#) &model)
Copy assignment.

- [CacheModel](#) & [operator+=](#) (const [CacheModel](#) &model)
Add the content of the given cache model.
- `template<typename T >`
`T & SerializeWith (T &stream)`
- [Codestream](#) & [GetCodestream](#) (int num_codestream)
Returns the reference of a codestream.
- int [GetMetadata](#) (int id)
Returns the amount of a meta-data.
- int [AddToMetadata](#) (int id, int amount, bool complete=false)
Increases the amount of a meta-data.
- `template<int BIN_CLASS>`
`int GetDataBin (int num_codestream, int id)`
Returns the amount of a data-bin item using the class [DataBinSelector](#).
- `template<int BIN_CLASS>`
`int AddToDataBin (int num_codestream, int id, int amount, bool complete=false)`
Increases the amount of a data-bin item using the class [DataBinSelector](#).
- bool [IsFullMetadata](#) () const
Returns the full flag of the meta-datas.
- void [SetFullMetadata](#) ()
Sets the full flag for the meta-datas to true.
- void [Pack](#) (int min_sum=1)
Calls the `Pack` method of all the codestreams.
- void [Clear](#) ()
Clear all the amounts.
- virtual [~CacheModel](#) ()

Private Attributes

- bool [full_meta](#)
Says if the meta-data has been totally sent.
- `vector< int > meta_data`
Amounts for the meta-datas.
- `vector< Codestream > codestreams`
Amounts for the codestreams.

6.8.1 Detailed Description

The cache model of a JPIP client is handled using this class.

It allows to maintain a cache model recording the amount of data sent by the server regarding the meta-datas, headers, tile-headers and precincts. This implementation only allows to record incremental amounts, from the beginning of each entity. The value `INT_MAX` is used to specify that an item is complete. This class is serializable.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 `jpip::CacheModel::CacheModel ()` `[inline]`

Empty constructor.

6.8.2.2 `jpip::CacheModel::CacheModel (const CacheModel & model)` `[inline]`

Copy constructor.

6.8.2.3 `virtual jpip::CacheModel::~~CacheModel () [inline], [virtual]`

6.8.3 Member Function Documentation

6.8.3.1 `template<int BIN_CLASS> int jpip::CacheModel::AddToDataBin (int num_codestream, int id, int amount, bool complete = false) [inline]`

Increases the amount of a data-bin item using the class [DataBinSelector](#).

Parameters

<i>num_↔ codestream</i>	Index number of the associated codestream.
<i>id</i>	Index number of the data-bin.
<i>amount</i>	Amount increment.
<i>complete</i>	true if the data-bin is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.8.3.2 `int jpip::CacheModel::AddToMetadata (int id, int amount, bool complete = false) [inline]`

Increases the amount of a meta-data.

Parameters

<i>id</i>	Index number of the meta-data.
<i>amount</i>	Amount increment.
<i>complete</i>	true if the meta-data is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.8.3.3 `void jpip::CacheModel::Clear () [inline]`

Clear all the amounts.

Here is the caller graph for this function:

6.8.3.4 `Codestream& jpip::CacheModel::GetCodestream (int num_codestream) [inline]`

Returns the reference of a codestream.

Parameters

<i>num_↔ codestream</i>	Index number of the codestream.
-----------------------------	---------------------------------

Here is the caller graph for this function:

6.8.3.5 `template<int BIN_CLASS> int jpip::CacheModel::GetDataBin (int num_codestream, int id) [inline]`

Returns the amount of a data-bin item using the class [DataBinSelector](#).

Parameters

<i>num_↔ codestream</i>	Index number of the associated codestream.
<i>id</i>	Index number of the data-bin.

Here is the caller graph for this function:

6.8.3.6 `int jpip::CacheModel::GetMetadata (int id) [inline]`

Returns the amount of a meta-data.

Parameters

<i>id</i>	Index number of the meta-data.
-----------	--------------------------------

Here is the caller graph for this function:

6.8.3.7 `bool jpip::CacheModel::IsFullMetadata () const [inline]`

Returns the full flag of the meta-datas.

Here is the caller graph for this function:

6.8.3.8 `CacheModel& jpip::CacheModel::operator+= (const CacheModel & model) [inline]`

Add the content of the given cache model.

6.8.3.9 `CacheModel& jpip::CacheModel::operator= (const CacheModel & model) [inline]`

Copy assignment.

Here is the call graph for this function:

6.8.3.10 `void jpip::CacheModel::Pack (int min_sum = 1) [inline]`

Calls the `Pack` method of all the codestreams.

Here is the caller graph for this function:

6.8.3.11 `template<typename T> T& jpip::CacheModel::SerializeWith (T & stream) [inline]`

6.8.3.12 `void jpip::CacheModel::SetFullMetadata () [inline]`

Sets the full flag for the meta-datas to true.

Here is the caller graph for this function:

6.8.4 Member Data Documentation

6.8.4.1 `vector<Codestream> jpip::CacheModel::codestreams [private]`

Amounts for the codestreams.

6.8.4.2 `bool jpip::CacheModel::full_meta` [private]

Says if the meta-data has been totally sent.

6.8.4.3 `vector<int> jpip::CacheModel::meta_data` [private]

Amounts for the meta-datas.

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

- [jpip/cache_model.h](#)

6.9 ClientInfo Class Reference

Contains information of a connected client.

```
#include <client_info.h>
```

Collaboration diagram for ClientInfo:

Public Member Functions

- [ClientInfo](#) (int [base_id](#), int [sock](#), int [father_sock](#))
Initializes the object.
- int [sock](#) () const
Returns the client socket.
- int [base_id](#) () const
Returns the base identifier.
- int [father_sock](#) () const
Returns the father socket.
- long [bytes_sent](#) () const
Returns the total bytes sent.
- long [time](#) () const
Returns the time spent from the starting of the connection.
- virtual [~ClientInfo](#) ()

Private Attributes

- int [sock_](#)
Client socket.
- int [base_id_](#)
Base identifier.
- time_t [tm_start](#)
When the connection started.
- int [father_sock_](#)
Father socket.
- long [bytes_sent_](#)
Total bytes sent.

6.9.1 Detailed Description

Contains information of a connected client.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 ClientInfo::ClientInfo (int *base_id*, int *sock*, int *father_sock*) [inline]

Initializes the object.

Parameters

<i>base_id</i>	Base identifier.
<i>sock</i>	Client socket.
<i>father_sock</i>	Father socket.

Here is the call graph for this function:

6.9.2.2 virtual ClientInfo::~~ClientInfo () [inline],[virtual]

6.9.3 Member Function Documentation

6.9.3.1 int ClientInfo::base_id () const [inline]

Returns the base identifier.

Here is the caller graph for this function:

6.9.3.2 long ClientInfo::bytes_sent () const [inline]

Returns the total bytes sent.

6.9.3.3 int ClientInfo::father_sock () const [inline]

Returns the father socket.

Here is the caller graph for this function:

6.9.3.4 int ClientInfo::sock () const [inline]

Returns the client socket.

Here is the caller graph for this function:

6.9.3.5 long ClientInfo::time () const [inline]

Returns the time spent from the starting of the connection.

Here is the caller graph for this function:

6.9.4 Member Data Documentation

6.9.4.1 int ClientInfo::base_id_ [private]

Base identifier.

6.9.4.2 long ClientInfo::bytes_sent_ [private]

Total bytes sent.

6.9.4.3 `int ClientInfo::father_sock_ [private]`

Father socket.

6.9.4.4 `int ClientInfo::sock_ [private]`

Client socket.

6.9.4.5 `time_t ClientInfo::tm_start [private]`

When the connection started.

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

- [client_info.h](#)

6.10 ClientManager Class Reference

Handles a client connection with a dedicated thread.

```
#include <client_manager.h>
```

Collaboration diagram for ClientManager:

Public Member Functions

- [ClientManager](#) ([AppConfig](#) &_cfg, [AppInfo](#) &_app_info, [IndexManager](#) &_index_manager)
Initializes the object.
- void [Run](#) ([ClientInfo](#) *client_info)
Starts the handling of a client connection.
- void [RunBasic](#) ([ClientInfo](#) *client_info)
Starts the handling of a client connection but it does not do anything.
- virtual [~ClientManager](#) ()

Private Attributes

- [AppConfig](#) & cfg
Application configuration.
- [AppInfo](#) & app_info
Application run-time information.
- [IndexManager](#) & index_manager
Index manager.

6.10.1 Detailed Description

Handles a client connection with a dedicated thread.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ClientManager::ClientManager (AppConfig & _cfg, AppInfo & _app_info, IndexManager & _index_manager) [inline]

Initializes the object.

Parameters

<code>_cfg</code>	Application configuration.
<code>_app_info</code>	Application run-time information.
<code>_index_manager</code>	Index manager.

6.10.2.2 `virtual ClientManager::~ClientManager () [inline],[virtual]`

6.10.3 Member Function Documentation

6.10.3.1 `void ClientManager::Run (ClientInfo * client_info)`

Starts the handling of a client connection.

Parameters

<code>client_info</code>	Client information.
--------------------------	---------------------

Here is the call graph for this function:

Here is the caller graph for this function:

6.10.3.2 `void ClientManager::RunBasic (ClientInfo * client_info)`

Starts the handling of a client connection but it does not do anything.

This method is used for testing the architecture of the server.

Parameters

<code>client_info</code>	Client information.
--------------------------	---------------------

Here is the call graph for this function:

Here is the caller graph for this function:

6.10.4 Member Data Documentation

6.10.4.1 `AppInfo& ClientManager::app_info [private]`

Application run-time information.

6.10.4.2 `AppConfig& ClientManager::cfg [private]`

Application configuration.

6.10.4.3 `IndexManager& ClientManager::index_manager [private]`

Index manager.

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

- [client_manager.h](#)
- [client_manager.cc](#)

6.11 jpip::CacheModel::Codestream Class Reference

Sub-class of the cache model class used to identify a codestream.

```
#include <cache_model.h>
```

Collaboration diagram for jpip::CacheModel::Codestream:

Public Member Functions

- [Codestream](#) ()
Initializes all the members to zero.
- [Codestream](#) (const [Codestream](#) &model)
Copy constructor.
- [Codestream](#) & [operator=](#) (const [Codestream](#) &model)
Copy assignment.
- [Codestream](#) & [operator+=](#) (const [Codestream](#) &model)
Add the content of the given codestream cache model.
- template<typename T >
T & [SerializeWith](#) (T &stream)
- int [GetMainHeader](#) () const
Returns the amount of the main header.
- int [GetTileHeader](#) () const
Returns the amount of the tile header.
- int [AddToMainHeader](#) (int amount, bool complete=false)
Increases the amount of the main header.
- int [AddToTileHeader](#) (int amount, bool complete=false)
Increases the amount of the tile header.
- int [GetPrecinct](#) (int num_precinct)
Returns the amount of a precinct.
- int [AddToPrecinct](#) (int num_precinct, int amount, bool complete=false)
Increases the amount of a precinct.
- void [Pack](#) (int min_sum=1)
Packs the information stored regarding the precincts, removing those initial elements that are consecutive and completes.

Private Attributes

- int [header](#)
Amount for the header.
- int [tile_header](#)
Amount for the tile-header.
- vector< int > [precincts](#)
Amount for the precincts.
- int [min_precinct](#)
Minimum identifier of the non-consecutive precinct completely sent.

6.11.1 Detailed Description

Sub-class of the cache model class used to identify a codestream.

This class is serializable.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 jpip::CacheModel::Codestream::Codestream () [inline]

Initializes all the members to zero.

6.11.2.2 jpip::CacheModel::Codestream::Codestream (const Codestream & model) [inline]

Copy constructor.

6.11.3 Member Function Documentation

6.11.3.1 int jpip::CacheModel::Codestream::AddToMainHeader (int amount, bool complete = false) [inline]

Increases the amount of the main header.

Parameters

<i>amount</i>	Amount increment.
<i>complete</i>	true if the main header is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.11.3.2 int jpip::CacheModel::Codestream::AddToPrecinct (int num_precinct, int amount, bool complete = false) [inline]

Increases the amount of a precinct.

Parameters

<i>num_precinct</i>	Index number of the precinct.
<i>amount</i>	Amount increment.
<i>complete</i>	true if the precinct is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.11.3.3 int jpip::CacheModel::Codestream::AddToTileHeader (int amount, bool complete = false) [inline]

Increases the amount of the tile header.

Parameters

<i>amount</i>	Amount increment.
<i>complete</i>	true if the tile header is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.11.3.4 `int jpip::CacheModel::Codestream::GetMainHeader () const [inline]`

Returns the amount of the main header.

Here is the caller graph for this function:

6.11.3.5 `int jpip::CacheModel::Codestream::GetPrecinct (int num_precinct) [inline]`

Returns the amount of a precinct.

Parameters

<i>num_precinct</i>	Index number of the precinct.
---------------------	-------------------------------

Here is the caller graph for this function:

6.11.3.6 `int jpip::CacheModel::Codestream::GetTileHeader () const [inline]`

Returns the amount of the tile header.

Here is the caller graph for this function:

6.11.3.7 `Codestream& jpip::CacheModel::Codestream::operator+= (const Codestream & model) [inline]`

Add the content of the given codestream cache model.

6.11.3.8 `Codestream& jpip::CacheModel::Codestream::operator= (const Codestream & model) [inline]`

Copy assignment.

Here is the call graph for this function:

6.11.3.9 `void jpip::CacheModel::Codestream::Pack (int min_sum = 1) [inline]`

Packs the information stored regarding the precincts, removing those initial elements that are consecutive and completes.

Parameters

<i>min_sum</i>	Only the packing is performed if there are a number of items equal or greater than this value (1 by default).
----------------	---

6.11.3.10 `template<typename T> T& jpip::CacheModel::Codestream::SerializeWith (T & stream) [inline]`

6.11.4 Member Data Documentation

6.11.4.1 `int jpip::CacheModel::Codestream::header [private]`

Amount for the header.

6.11.4.2 `int jpip::CacheModel::Codestream::min_precinct [private]`

Minimum identifier of the non-consecutive precinct completely sent.

All the initial precincts already sent completely to the client are removed, so this value contains the next precinct. The vector `precincts` is related to the precincts starting from this index.

6.11.4.3 `vector<int> jpip::CacheModel::Codestream::precincts` `[private]`

Amount for the precincts.

6.11.4.4 `int jpip::CacheModel::Codestream::tile_header` `[private]`

Amount for the tile-header.

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

- [jpip/cache_model.h](#)

6.12 jpeg2000::CodestreamIndex Class Reference

Class used for indexing the information of a JPEG2000 codestream.

```
#include <codestream_index.h>
```

Collaboration diagram for jpeg2000::CodestreamIndex:

Public Member Functions

- [CodestreamIndex](#) ()
Empty constructor.
- [CodestreamIndex](#) (const [CodestreamIndex](#) &index)
Copy constructor.
- void [Clear](#) ()
Clears the information.
- const [CodestreamIndex](#) & [operator=](#) (const [CodestreamIndex](#) &index)
Copy assignment.
- template<typename T >
T & [SerializeWith](#) (T &stream)
- virtual [~CodestreamIndex](#) ()

Public Attributes

- [FileSegment](#) header
Main header segment.
- vector< [FileSegment](#) > packets
Tile-part packets segments.
- vector< [FileSegment](#) > PLT_markers
PLT markers segments.

Friends

- ostream & [operator<<](#) (ostream &out, const [CodestreamIndex](#) &index)

6.12.1 Detailed Description

Class used for indexing the information of a JPEG2000 codestream.

The indexed information is the segment of the main header, the contiguous segments of packets (usually the data of each tile-part) and the segments of the existing PLT markers. This class can be printed and serialized.

See also

[data::FileSegment](#)

6.12.2 Constructor & Destructor Documentation

6.12.2.1 `jpeg2000::CodestreamIndex::CodestreamIndex ()` `[inline]`

Empty constructor.

6.12.2.2 `jpeg2000::CodestreamIndex::CodestreamIndex (const CodestreamIndex & index)` `[inline]`

Copy constructor.

6.12.2.3 `virtual jpeg2000::CodestreamIndex::~CodestreamIndex ()` `[inline],[virtual]`

6.12.3 Member Function Documentation

6.12.3.1 `void jpeg2000::CodestreamIndex::Clear ()` `[inline]`

Clears the information.

6.12.3.2 `const CodestreamIndex& jpeg2000::CodestreamIndex::operator= (const CodestreamIndex & index)`
`[inline]`

Copy assignment.

Here is the call graph for this function:

6.12.3.3 `template<typename T> T& jpeg2000::CodestreamIndex::SerializeWith (T & stream)` `[inline]`

6.12.4 Friends And Related Function Documentation

6.12.4.1 `ostream& operator<< (ostream & out, const CodestreamIndex & index)` `[friend]`

6.12.5 Member Data Documentation

6.12.5.1 `FileSegment jpeg2000::CodestreamIndex::header`

Main header segment.

6.12.5.2 `vector<FileSegment> jpeg2000::CodestreamIndex::packets`

Tile-part packets segments.

6.12.5.3 vector<FileSegment> jpeg2000::CodestreamIndex::PLT_markers

PLT markers segments.

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

- jpeg2000/codestream_index.h

6.13 jpeg2000::CodingParameters Class Reference

Contains the coding parameters of a JPEG2000 image codestream.

```
#include <coding_parameters.h>
```

Collaboration diagram for jpeg2000::CodingParameters:

Public Types

- enum {
[LRCP_PROGRESSION](#) = 0, [RLCP_PROGRESSION](#) = 1, [RPCL_PROGRESSION](#) = 2, [PCRL_PROGRESSION](#) = 3,
[CPRL_PROGRESSION](#) = 4 }
All the progression orders defined in the JPEG2000 standard (Part 1).
- typedef [SHARED_PTR](#)< [CodingParameters](#) > [Ptr](#)
Pointer to an object of this class.

Public Member Functions

- [CodingParameters](#) ()
Initializes the object.
- [CodingParameters](#) (const [CodingParameters](#) &cod_params)
Copy constructor.
- const [CodingParameters](#) & [operator=](#) (const [CodingParameters](#) &cod_params)
Copy assignment.
- template<typename T >
[T](#) & [SerializeWith](#) ([T](#) &stream)
- bool [IsResolutionProgression](#) () const
Returns true if the progression is RLCP or RPCL.
- [Size](#) [GetPrecincts](#) (int r, const [Size](#) &point)
Returns a precinct coordinate adjusted to a given resolution level.
- int [GetProgressionIndex](#) (const [Packet](#) &packet)
Returns the index of a packet according to the progression order.
- int [GetPrecinctDataBinId](#) (const [Packet](#) &packet)
Returns the data-bin identifier associated to the given packet.
- int [GetClosestResolution](#) (const [Size](#) &res_size, [Size](#) *res_image_size)
Returns the resolution level according to the given size and the closest round policy.
- int [GetRoundUpResolution](#) (const [Size](#) &res_size, [Size](#) *res_image_size)
Returns the resolution level according to the given size and the round-up round policy.
- int [GetRoundDownResolution](#) (const [Size](#) &res_size, [Size](#) *res_image_size)
Returns the resolution level according to the given size and the round-down round policy.
- virtual [~CodingParameters](#) ()

Public Attributes

- [Size](#) `size`
Image size.
- `int` [num_levels](#)
Number of resolution levels.
- `int` [num_layers](#)
Number of quality layers.
- `int` [progression](#)
Progression order.
- `int` [num_components](#)
Number of components.
- `vector< Size >` [precinct_size](#)
Precinct sizes of each resolution level.

Private Member Functions

- `void` [FillTotalPrecinctsVector](#) ()
Fills the vector `total_precincts`.
- `int` [GetProgressionIndexRPCL](#) (`int` l, `int` r, `int` c, `int` px, `int` py)
Returns the index of a packet according to the RPCL progression.
- `int` [GetProgressionIndexRLCP](#) (`int` l, `int` r, `int` c, `int` px, `int` py)
Returns the index of a packet according to the RLCP progression.
- `int` [GetProgressionIndexLRCP](#) (`int` l, `int` r, `int` c, `int` px, `int` py)
Returns the index of a packet according to the LRCP progression.

Private Attributes

- `vector< int >` [total_precincts](#)
Contains the number of precincts of each resolution level.

Friends

- `ostream` & [operator<<](#) (`ostream` &out, `const` [CodingParameters](#) ¶ms)

6.13.1 Detailed Description

Contains the coding parameters of a JPEG2000 image codestream.

This class can be serialized and printed.

6.13.2 Member Typedef Documentation

6.13.2.1 `typedef SHARED_PTR<CodingParameters> jpeg2000::CodingParameters::Ptr`

Pointer to an object of this class.

6.13.3 Member Enumeration Documentation

6.13.3.1 anonymous enum

All the progression orders defined in the JPEG2000 standard (Part 1).

Enumerator

LRCP_PROGRESSION LRCP.
RLCP_PROGRESSION RLCP.
RPCL_PROGRESSION RPCL.
PCRL_PROGRESSION PCRL.
CPRL_PROGRESSION CPRL.

6.13.4 Constructor & Destructor Documentation

6.13.4.1 jpeg2000::CodingParameters::CodingParameters () [inline]

Initializes the object.

6.13.4.2 jpeg2000::CodingParameters::CodingParameters (const CodingParameters & *cod_params*) [inline]

Copy constructor.

6.13.4.3 virtual jpeg2000::CodingParameters::~~CodingParameters () [inline],[virtual]

6.13.5 Member Function Documentation

6.13.5.1 void jpeg2000::CodingParameters::FillTotalPrecinctsVector () [private]

Fills the vector `total_precincts`.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.2 int jpeg2000::CodingParameters::GetClosestResolution (const Size & *res_size*, Size * *res_image_size*)

Returns the resolution level according to the given size and the closest round policy.

Parameters

<i>res_size</i>	Resolution size.
<i>res_image_size</i>	Image size associated to the resolution level returned.

6.13.5.3 int jpeg2000::CodingParameters::GetPrecinctDataBinId (const Packet & *packet*) [inline]

Returns the data-bin identifier associated to the given packet.

Parameters

<i>packet</i>	Packet information.
---------------	-------------------------------------

Here is the call graph for this function:

6.13.5.4 `Size jpeg2000::CodingParameters::GetPrecincts (int r, const Size & point) [inline]`

Returns a precinct coordinate adjusted to a given resolution level.

Parameters

<i>r</i>	Resolution level.
<i>point</i>	Precinct coordinate.

Here is the caller graph for this function:

6.13.5.5 `int jpeg2000::CodingParameters::GetProgressionIndex (const Packet & packet) [inline]`

Returns the index of a packet according to the progression order.

Parameters

<i>packet</i>	Packet information.
---------------	-------------------------------------

Here is the call graph for this function:

6.13.5.6 `int jpeg2000::CodingParameters::GetProgressionIndexLRCP (int l, int r, int c, int px, int py) [inline], [private]`

Returns the index of a packet according to the LRCP progression.

Parameters

<i>l</i>	Quality layer.
<i>r</i>	Resolution level.
<i>c</i>	Component.
<i>px</i>	Precinct position X.
<i>py</i>	Precinct position Y.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.7 `int jpeg2000::CodingParameters::GetProgressionIndexRLCP (int l, int r, int c, int px, int py) [inline], [private]`

Returns the index of a packet according to the RLCP progression.

Parameters

<i>l</i>	Quality layer.
<i>r</i>	Resolution level.
<i>c</i>	Component.
<i>px</i>	Precinct position X.
<i>py</i>	Precinct position Y.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.8 `int jpeg2000::CodingParameters::GetProgressionIndexRPCL (int l, int r, int c, int px, int py)` `[inline]`,
`[private]`

Returns the index of a packet according to the RPCL progression.

Parameters

<i>l</i>	Quality layer.
<i>r</i>	Resolution level.
<i>c</i>	Component.
<i>px</i>	Precinct position X.
<i>py</i>	Precinct position Y.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.9 `int jpeg2000::CodingParameters::GetRoundDownResolution (const Size & res_size, Size * res_image_size)`

Returns the resolution level according to the given size and the round-down round policy.

Parameters

<i>res_size</i>	Resolution size.
<i>res_image_size</i>	Image size associated to the resolution level returned.

6.13.5.10 `int jpeg2000::CodingParameters::GetRoundUpResolution (const Size & res_size, Size * res_image_size)`

Returns the resolution level according to the given size and the round-up round policy.

Parameters

<i>res_size</i>	Resolution size.
<i>res_image_size</i>	Image size associated to the resolution level returned.

6.13.5.11 `bool jpeg2000::CodingParameters::IsResolutionProgression () const [inline]`

Returns `true` if the progression is RLCP or RPCL.

6.13.5.12 `const CodingParameters& jpeg2000::CodingParameters::operator= (const CodingParameters & cod_params) [inline]`

Copy assignment.

Here is the call graph for this function:

6.13.5.13 `template<typename T> T& jpeg2000::CodingParameters::SerializeWith (T & stream) [inline]`

6.13.6 Friends And Related Function Documentation

6.13.6.1 `ostream& operator<< (ostream & out, const CodingParameters & params) [friend]`

6.13.7 Member Data Documentation

6.13.7.1 `int jpeg2000::CodingParameters::num_components`

Number of components.

6.13.7.2 int jpeg2000::CodingParameters::num_layers

Number of quality layers.

6.13.7.3 int jpeg2000::CodingParameters::num_levels

Number of resolution levels.

6.13.7.4 vector<Size> jpeg2000::CodingParameters::precinct_size

Precinct sizes of each resolution level.

6.13.7.5 int jpeg2000::CodingParameters::progression

Progression order.

6.13.7.6 Size jpeg2000::CodingParameters::size

Image size.

6.13.7.7 vector<int> jpeg2000::CodingParameters::total_precincts [private]

Contains the number of precincts of each resolution level.

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

- [jpeg2000/coding_parameters.h](#)
- [jpeg2000/coding_parameters.cc](#)

6.14 ApplInfo::Data Struct Reference

Contains the data block that is maintained in shared memory.

Collaboration diagram for ApplInfo::Data:

Public Member Functions

- void [Reset](#) ()
Clears the values.

Public Attributes

- int [father_pid](#)
PID of the father process.
- int [child_pid](#)
PID of the child process.
- int [num_connections](#)
Number of open connections.
- int [child_iterations](#)
Number of iterations done by the child.

6.14.1 Detailed Description

Contains the data block that is maintained in shared memory.

6.14.2 Member Function Documentation

6.14.2.1 `void AppInfo::Data::Reset()` `[inline]`

Clears the values.

6.14.3 Member Data Documentation

6.14.3.1 `int AppInfo::Data::child_iterations`

Number of iterations done by the child.

6.14.3.2 `int AppInfo::Data::child_pid`

PID of the child process.

6.14.3.3 `int AppInfo::Data::father_pid`

PID of the father process.

6.14.3.4 `int AppInfo::Data::num_connections`

Number of open connections.

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

- [app_info.h](#)

6.15 jpip::DataBinClass Class Reference

Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.

```
#include <jpip.h>
```

Collaboration diagram for jpip::DataBinClass:

Public Types

- enum {
`PRECINCT` = 0, `EXTENDED_PRECINCT` = 1, `TILE_HEADER` = 2, `TILE_DATA` = 4,
`EXTENDED_TILE` = 5, `MAIN_HEADER` = 6, `META_DATA` = 8 }

Static Public Member Functions

- static const char * `GetName` (int class_name)
Returns a string with the name of the databin class name given.

Private Member Functions

- [DataBinClass](#) ()

6.15.1 Detailed Description

Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.

It is not possible to create an object of this class.

6.15.2 Member Enumeration Documentation

6.15.2.1 anonymous enum

Enumerator

PRECINCT Class identifier for precinct data-bins.

EXTENDED_PRECINCT Class identifier for extended precinct data-bins.

TILE_HEADER Class identifier for tile header data-bins.

TILE_DATA Class identifier for tile data-bins.

EXTENDED_TILE Class identifier for extended tile data-bins.

MAIN_HEADER Class identifier for main header data-bins.

META_DATA Class identifier for meta-data data-bins.

6.15.3 Constructor & Destructor Documentation

6.15.3.1 jpip::DataBinClass::DataBinClass () [inline], [private]

6.15.4 Member Function Documentation

6.15.4.1 const char * jpip::DataBinClass::GetName (int class_name) [static]

Returns a string with the name of the databin class name given,.

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

- [jpip/jpip.h](#)
- [jpip/jpip.cc](#)

6.16 jpip::DataBinSelector< BIN_CLASS > Struct Template Reference

Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.

```
#include <cache_model.h>
```

Collaboration diagram for jpip::DataBinSelector< BIN_CLASS >:

6.16.1 Detailed Description

```
template<int BIN_CLASS>struct jpip::DataBinSelector< BIN_CLASS >
```

Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.

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

- [jpip/cache_model.h](#)

6.17 jpip::DataBinSelector< DataBinClass::MAIN_HEADER > Struct Template Reference

```
#include <cache_model.h>
```

Collaboration diagram for jpip::DataBinSelector< DataBinClass::MAIN_HEADER >:

Static Public Member Functions

- static int [Get](#) ([CacheModel](#) &model, int num_codestream, int id)
- static int [AddTo](#) ([CacheModel](#) &model, int num_codestream, int id, int amount, bool complete)

6.17.1 Member Function Documentation

6.17.1.1 static int jpip::DataBinSelector< DataBinClass::MAIN_HEADER >::AddTo ([CacheModel](#) & *model*, int *num_codestream*, int *id*, int *amount*, bool *complete*) `[inline]`, `[static]`

Here is the call graph for this function:

6.17.1.2 static int jpip::DataBinSelector< DataBinClass::MAIN_HEADER >::Get ([CacheModel](#) & *model*, int *num_codestream*, int *id*) `[inline]`, `[static]`

Here is the call graph for this function:

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

- [jpip/cache_model.h](#)

6.18 jpip::DataBinSelector< DataBinClass::META_DATA > Struct Template Reference

```
#include <cache_model.h>
```

Collaboration diagram for jpip::DataBinSelector< DataBinClass::META_DATA >:

Static Public Member Functions

- static int [Get](#) ([CacheModel](#) &model, int num_codestream, int id)
- static int [AddTo](#) ([CacheModel](#) &model, int num_codestream, int id, int amount, bool complete)

6.18.1 Member Function Documentation

6.18.1.1 static int jpip::DataBinSelector< DataBinClass::META_DATA >::AddTo ([CacheModel](#) & *model*, int *num_codestream*, int *id*, int *amount*, bool *complete*) `[inline]`, `[static]`

Here is the call graph for this function:

6.18.1.2 `static int jpip::DataBinSelector< DataBinClass::META_DATA >::Get (CacheModel & model, int num_codestream, int id) [inline], [static]`

Here is the call graph for this function:

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

- [jpip/cache_model.h](#)

6.19 jpip::DataBinSelector< DataBinClass::PRECINCT > Struct Template Reference

```
#include <cache_model.h>
```

Collaboration diagram for jpip::DataBinSelector< DataBinClass::PRECINCT >:

Static Public Member Functions

- static int [Get](#) ([CacheModel](#) &model, int num_codestream, int id)
- static int [AddTo](#) ([CacheModel](#) &model, int num_codestream, int id, int amount, bool complete)

6.19.1 Member Function Documentation

6.19.1.1 `static int jpip::DataBinSelector< DataBinClass::PRECINCT >::AddTo (CacheModel & model, int num_codestream, int id, int amount, bool complete) [inline], [static]`

Here is the call graph for this function:

6.19.1.2 `static int jpip::DataBinSelector< DataBinClass::PRECINCT >::Get (CacheModel & model, int num_codestream, int id) [inline], [static]`

Here is the call graph for this function:

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

- [jpip/cache_model.h](#)

6.20 jpip::DataBinSelector< DataBinClass::TILE_HEADER > Struct Template Reference

```
#include <cache_model.h>
```

Collaboration diagram for jpip::DataBinSelector< DataBinClass::TILE_HEADER >:

Static Public Member Functions

- static int [Get](#) ([CacheModel](#) &model, int num_codestream, int id)
- static int [AddTo](#) ([CacheModel](#) &model, int num_codestream, int id, int amount, bool complete)

6.20.1 Member Function Documentation

6.20.1.1 `static int jpip::DataBinSelector< DataBinClass::TILE_HEADER >::AddTo (CacheModel & model, int num_codestream, int id, int amount, bool complete) [inline], [static]`

Here is the call graph for this function:

6.20.1.2 `static int jpip::DataBinSelector< DataBinClass::TILE_HEADER >::Get (CacheModel & model, int num_codestream, int id) [inline], [static]`

Here is the call graph for this function:

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

- [jpip/cache_model.h](#)

6.21 jpip::DataBinServer Class Reference

Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.

```
#include <databin_server.h>
```

Collaboration diagram for jpip::DataBinServer:

Public Member Functions

- [DataBinServer](#) ()
Initializes the object.
- bool [end_woi](#) () const
Returns `true` if the end of the `WOI` has been reached, that is, there is not more associated packets to send.
- bool [Reset](#) (const [ImageIndex::Ptr](#) image_index)
Resets the server assigning a new image to serve.
- bool [SetRequest](#) (const [Request](#) &req)
Sets the new current request to take into account for generating the chunks of data.
- bool [GenerateChunk](#) (char *buff, int *len, bool *last)
Generates a new chunk of data for the current image and `WOI`, according to the last indicated request.
- virtual [~DataBinServer](#) ()

Private Types

- enum { [MINIMUM_SPACE](#) = 60 }

Private Member Functions

- template<int BIN_CLASS>
int [WriteSegment](#) (int num_codestream, int id, [FileSegment](#) segment, int offset=0, bool last=true)
Writes a new data-bin segment or a part of it that is not already cached.
- int [WritePlaceholder](#) (int num_codestream, int id, const [Placeholder](#) &place_holder, int offset=0, bool last=false)
Writes a new place-holder segment, only if it is possible to write it completely.

Private Attributes

- [WOI](#) woi
Current `WOI`.
- int [pending](#)
Number of pending bytes.
- [Range](#) range

- Range of codestreams.*
 - bool `has_woi`
true if the last request contained a WOI
 - bool `metareq`
true if the last request contained a "metareq"
 - bool `end_woi_`
true if the WOI has been completely sent
 - `File::Ptr` `file`
Pointer to the associated image file.
 - int `current_idx`
Current codestream index.
 - bool `eof`
true if the end has been reached and the last write operation could not be completed.
 - `CacheModel` `cache_model`
Cache model of the client.
 - `vector< File::Ptr >` `files`
List of files (for hyperlinked JPX files)
 - `WOIComposer` `woi_composer`
WOI composer for determining the packets.
 - `ImageIndex::Ptr` `im_index`
Pointer to the associated image index.
 - `DataBinWriter` `data_writer`
Data-bin writer for generating the chunks.

6.21.1 Detailed Description

Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.

6.21.2 Member Enumeration Documentation

6.21.2.1 anonymous enum [private]

Enumerator

MINIMUM_SPACE Minimum space in the chunk.

6.21.3 Constructor & Destructor Documentation

6.21.3.1 `jpip::DataBinServer::DataBinServer ()` [inline]

Initializes the object.

6.21.3.2 `virtual jpip::DataBinServer::~~DataBinServer ()` [inline], [virtual]

6.21.4 Member Function Documentation

6.21.4.1 `bool jpip::DataBinServer::end_woi () const` [inline]

Returns `true` if the end of the `WOI` has been reached, that is, there is not more associated packets to send.

Here is the caller graph for this function:

6.21.4.2 `bool jpip::DataBinServer::GenerateChunk (char * buff, int * len, bool * last)`

Generates a new chunk of data for the current image and [WOI](#), according to the last indicated request.

Parameters

<i>buff</i>	Pointer to the memory buffer.
<i>len</i>	Length of the memory buffer. It is modified by the method to indicate how many bytes have been written to the buffer.
<i>last</i>	Output parameter to indicates if this is the last chunk of data associated to the last request.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.21.4.3 `bool jpip::DataBinServer::Reset (const ImageIndex::Ptr image_index)`

Resets the server assigning a new image to serve.

It also resets the maintained cache model.

Parameters

<i>image_index</i>	Pointer to the new image index to use.
--------------------	--

Returns

`true` if successful.

Here is the caller graph for this function:

6.21.4.4 `bool jpip::DataBinServer::SetRequest (const Request & req)`

Sets the new current request to take into account for generating the chunks of data.

Parameters

<i>req</i>	Request .
------------	---------------------------

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.21.4.5 `int jpip::DataBinServer::WritePlaceholder (int num_codestream, int id, const Placeholder & place_holder, int offset = 0, bool last = false) [inline],[private]`

Writes a new place-holder segment, only if it is possible to write it completely.

Parameters

<i>num_codestream</i>	Index number of the codestream.
-----------------------	---------------------------------

<i>id</i>	Data-bin identifier.
<i>place_holder</i>	Place-holder information.
<i>offset</i>	Data-bin offset of the data (0 by default).
<i>last</i>	<code>true</code> if this is the last data of the data-bin.

Returns

1 if the segment content was completely written and/or cached, 0 if it was incompletely written (or not at all, if EOF flag is set), or -1 if an error was generated.

Here is the call graph for this function:

Here is the caller graph for this function:

6.21.4.6 `template<int BIN_CLASS> int jpip::DataBinServer::WriteSegment (int num_codestream, int id, FileSegment segment, int offset = 0, bool last = true) [inline], [private]`

Writes a new data-bin segment or a part of it that is not already cached.

Parameters

<i>num_↔ codestream</i>	Index number of the codestream.
<i>id</i>	Data-bin identifier.
<i>segment</i>	File segment associated.
<i>offset</i>	Data-bin offset of the data (0 by default).
<i>last</i>	<code>true</code> if this is the last data of the data-bin.

Returns

1 if the segment content was completely written and/or cached, 0 if it was incompletely written (or not at all, if EOF flag is set), or -1 if an error was generated.

Here is the call graph for this function:

6.21.5 Member Data Documentation

6.21.5.1 `CacheModel jpip::DataBinServer::cache_model [private]`

Cache model of the client.

6.21.5.2 `int jpip::DataBinServer::current_idx [private]`

Current codestream index.

6.21.5.3 `DataBinWriter jpip::DataBinServer::data_writer [private]`

Data-bin writer for generating the chunks.

6.21.5.4 `bool jpip::DataBinServer::end_woi_ [private]`

`true` if the [WOI](#) has been completely sent

6.21.5.5 `bool jpip::DataBinServer::eof` `[private]`

`true` if the end has been reached and the last write operation could not be completed.

6.21.5.6 `File::Ptr jpip::DataBinServer::file` `[private]`

Pointer to the associated image file.

6.21.5.7 `vector<File::Ptr> jpip::DataBinServer::files` `[private]`

List of files (for hyperlinked JPX files)

6.21.5.8 `bool jpip::DataBinServer::has_woi` `[private]`

`true` if the last request contained a [WOI](#)

6.21.5.9 `ImageIndex::Ptr jpip::DataBinServer::im_index` `[private]`

Pointer to the associated image index.

6.21.5.10 `bool jpip::DataBinServer::metareq` `[private]`

`true` if the last request contained a "metareq"

6.21.5.11 `int jpip::DataBinServer::pending` `[private]`

Number of pending bytes.

6.21.5.12 `Range jpip::DataBinServer::range` `[private]`

Range of codestreams.

6.21.5.13 `WOI jpip::DataBinServer::woi` `[private]`

Current [WOI](#).

6.21.5.14 `WOIComposer jpip::DataBinServer::woi_composer` `[private]`

[WOI](#) composer for determining the packets.

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

- [jpip/databin_server.h](#)
- [jpip/databin_server.cc](#)

6.22 jpip::DataBinWriter Class Reference

Class used to generate data-bin segments and write them into a memory buffer.

```
#include <databin_writer.h>
```

Collaboration diagram for jpip::DataBinWriter:

Public Member Functions

- [DataBinWriter](#) ()
Initializes the object.
- [DataBinWriter](#) & [SetBuffer](#) (char *buf, int buf_len)
Sets the associated memory buffer.
- [DataBinWriter](#) & [ClearPreviousIds](#) ()
Clears the previous identifiers of data-bin class and codestream index numbers.
- [DataBinWriter](#) & [SetCodestream](#) (int value)
Sets the current codestream.
- [DataBinWriter](#) & [SetDataBinClass](#) (int databin_class)
Sets the current data-bin class.
- [DataBinWriter](#) & [Write](#) (uint64_t bin_id, uint64_t bin_offset, const [File](#) &file, const [FileSegment](#) &segment, bool last_byte=false)
Writes a data-bin segment into the buffer.
- [DataBinWriter](#) & [WritePlaceholder](#) (uint64_t bin_id, uint64_t bin_offset, const [File](#) &file, const [Placeholder](#) &place_holder, bool last_byte=false)
Writes a place-holder segment into the buffer.
- [DataBinWriter](#) & [WriteEmpty](#) (uint64_t bin_id=0)
Writes an empty segment.
- int [GetCount](#) () const
Returns the number of bytes written.
- int [GetFree](#) () const
Returns the number of bytes available.
- [DataBinWriter](#) & [WriteEOR](#) (int reason)
*Writes a *EOR* message into the buffer.*
- operator bool () const
*Returns the *EOF* status of the object.*
- virtual [~DataBinWriter](#) ()

Private Member Functions

- template<typename T >
[DataBinWriter](#) & [WriteValue](#) (T value)
Writes a value into the buffer.
- [DataBinWriter](#) & [WriteVBAS](#) (uint64_t value)
Writes a new integer value into the buffer coded as VBAS.
- [DataBinWriter](#) & [WriteHeader](#) (uint64_t bin_id, uint64_t bin_offset, uint64_t bin_length, bool last_byte=false)
Writes a data-bin header into the buffer.

Private Attributes

- bool eof
true if the end of the buffer has been reached and the last value could not be written.
- char * ini
Pointer to the beginning of the buffer.
- char * ptr
Current position of the buffer.
- char * end
Pointer to the end of the buffer.
- int databin_class

- Current data-bin class.*
- int [codestream_idx](#)
Current codestream index number.
- int [prev_databin_class](#)
Previous data-bin class.
- int [prev_codestream_idx](#)
Previous codestream index number.

6.22.1 Detailed Description

Class used to generate data-bin segments and write them into a memory buffer.

See also

[DataBinServer](#)
[DataBinClass](#)
[EOR](#)

6.22.2 Constructor & Destructor Documentation

6.22.2.1 jpip::DataBinWriter::DataBinWriter () [inline]

Initializes the object.

6.22.2.2 virtual jpip::DataBinWriter::~~DataBinWriter () [inline], [virtual]

6.22.3 Member Function Documentation

6.22.3.1 DataBinWriter& jpip::DataBinWriter::ClearPreviousIds () [inline]

Clears the previous identifiers of data-bin class and codestream index numbers.

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.2 int jpip::DataBinWriter::GetCount () const [inline]

Returns the number of bytes written.

Here is the caller graph for this function:

6.22.3.3 int jpip::DataBinWriter::GetFree () const [inline]

Returns the number of bytes available.

Here is the caller graph for this function:

6.22.3.4 jpip::DataBinWriter::operator bool () const [inline]

Returns the EOF status of the object.

6.22.3.5 DataBinWriter& jpip::DataBinWriter::SetBuffer (char * *buf*, int *buf_len*) [inline]

Sets the associated memory buffer.

Parameters

<i>buf</i>	Memory buffer.
<i>buf_len</i>	Length of the memory buffer.

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.6 DataBinWriter& jpip::DataBinWriter::SetCodestream (int *value*) [inline]

Sets the current codestream.

Parameters

<i>value</i>	Index number of the codestream.
--------------	---------------------------------

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.7 DataBinWriter& jpip::DataBinWriter::SetDataBinClass (int *databin_class*) [inline]

Sets the current data-bin class.

Parameters

<i>databin_class</i>	Data-bin class.
----------------------	-----------------

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.8 DataBinWriter & jpip::DataBinWriter::Write (uint64_t *bin_id*, uint64_t *bin_offset*, const File & *file*, const FileSegment & *segment*, bool *last_byte* = false)

Writes a data-bin segment into the buffer.

Parameters

<i>bin_id</i>	Data-bin identifier.
<i>bin_offset</i>	Data-bin offset.
<i>file</i>	File from where to read the data.
<i>segment</i>	File segment of the data.
<i>last_byte</i>	true if the data contains the last byte of the data-bin.

Returns

The object itself.

Here is the call graph for this function:

Here is the caller graph for this function:

6.22.3.9 DataBinWriter & jpip::DataBinWriter::WriteEmpty (uint64_t *bin_id* = 0)

Writes an empty segment.

Parameters

<i>bin_id</i>	Data-bin identifier.
---------------	----------------------

Returns

The object itself.

Here is the call graph for this function:

6.22.3.10 DataBinWriter& jpip::DataBinWriter::WriteEOR (int *reason*) [inline]

Writes a [EOR](#) message into the buffer.

Parameters

<i>reason</i>	Reason of the message.
---------------	------------------------

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.11 DataBinWriter & jpip::DataBinWriter::WriteHeader (uint64_t *bin_id*, uint64_t *bin_offset*, uint64_t *bin_length*, bool *last_byte = false*) [private]

Writes a data-bin header into the buffer.

Parameters

<i>bin_id</i>	Data-bin identifier.
<i>bin_offset</i>	Data-bin offset.
<i>bin_length</i>	Data-bin length.
<i>last_byte</i>	<code>true</code> if the data related to this header contains the last byte of the data-bin.

Returns

The object itself.

Here is the call graph for this function:

Here is the caller graph for this function:

6.22.3.12 DataBinWriter & jpip::DataBinWriter::WritePlaceholder (uint64_t *bin_id*, uint64_t *bin_offset*, const File & *file*, const Placeholder & *place_holder*, bool *last_byte = false*)

Writes a place-holder segment into the buffer.

Parameters

<i>bin_id</i>	Data-bin identifier.
<i>bin_offset</i>	Data-bin offset.
<i>file</i>	File from where to read the data.

<i>place_holder</i>	Place-holder information.
<i>last_byte</i>	true if the data contains the last byte of the data-bin.

Returns

The object itself.

Here is the call graph for this function:

Here is the caller graph for this function:

6.22.3.13 `template<typename T > DataBinWriter& jpip::DataBinWriter::WriteValue (T value) [inline],
[private]`

Writes a value into the buffer.

Parameters

<i>value</i>	Value to write.
--------------	-----------------

Returns

The object itself.

6.22.3.14 `DataBinWriter & jpip::DataBinWriter::WriteVBAS (uint64_t value) [private]`

Writes a new integer value into the buffer coded as VBAS.

Parameters

<i>value</i>	Value to write.
--------------	-----------------

Returns

The object itself.

Here is the caller graph for this function:

6.22.4 Member Data Documentation

6.22.4.1 `int jpip::DataBinWriter::codestream_idx [private]`

Current codestream index number.

6.22.4.2 `int jpip::DataBinWriter::databin_class [private]`

Current data-bin class.

6.22.4.3 `char* jpip::DataBinWriter::end [private]`

Pointer to the end of the buffer.

6.22.4.4 `bool jpip::DataBinWriter::eof [private]`

true if the end of the buffer has been reached and the last value could not be written.

6.22.4.5 `char* jpip::DataBinWriter::ini` [private]

Pointer to the beginning of the buffer.

6.22.4.6 `int jpip::DataBinWriter::prev_codestream_idx` [private]

Previous codestream index number.

6.22.4.7 `int jpip::DataBinWriter::prev_databin_class` [private]

Previous data-bin class.

6.22.4.8 `char* jpip::DataBinWriter::ptr` [private]

Current position of the buffer.

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

- [jpip/databin_writer.h](#)
- [jpip/databin_writer.cc](#)

6.23 jpip::EOR Class Reference

Class that contains all the definitions of the EOF messages defined for the JPIP protocol.

```
#include <jpip.h>
```

Collaboration diagram for jpip::EOR:

Public Types

- enum {
`IMAGE_DONE = 1, WINDOW_DONE = 2, WINDOW_CHANGE = 3, BYTE_LIMIT_REACHED = 4,`
`QUALITY_LIMIT_REACHED = 5, SESSION_LIMIT_REACHED = 6, RESPONSE_LIMIT_REACHED = 7,`
`NON_SPECIFIED = 0xFF }`

Private Member Functions

- [EOR\(\)](#)

6.23.1 Detailed Description

Class that contains all the definitions of the EOF messages defined for the JPIP protocol.

It is not possible to create an object of this class.

6.23.2 Member Enumeration Documentation

6.23.2.1 anonymous enum

Enumerator

IMAGE_DONE [EOR](#) code sent when the server has transferred all available image information (not just information relevant to the requested view-window) to the client.

WINDOW_DONE [EOR](#) code sent when the server has transferred all available information that is relevant to the requested view-window.

WINDOW_CHANGE [EOR](#) code sent when the server is terminating its response in order to service a new request.

BYTE_LIMIT_REACHED [EOR](#) code sent when the server is terminating its response because the byte limit specified in a byte limit specified in a max length request field has been reached.

QUALITY_LIMIT_REACHED [EOR](#) code sent when the server is terminating its response because the quality limit specified in a quality request field has been reached.

SESSION_LIMIT_REACHED [EOR](#) code sent when the server is terminating its response because some limit on the session resources, e.g. a time limit, has been reached. No further request should be issued using a channel ID assigned in that session.

RESPONSE_LIMIT_REACHED [EOR](#) code sent when the server is terminating its response because some limit, e.g., a time limit, has been reached. If the request is issued in a session, further requests can still be issued using a channel ID assigned in that session.

NON_SPECIFIED [EOR](#) code sent when there is not any specific [EOR](#) reason.

6.23.3 Constructor & Destructor Documentation

6.23.3.1 `jpeg::EOR::EOR()` `[inline]`, `[private]`

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

- [jpeg/jpeg.h](#)

6.24 ipc::Event Class Reference

IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API.

```
#include <event.h>
```

Inheritance diagram for `ipc::Event`:

Collaboration diagram for `ipc::Event`:

Public Types

- typedef [SHARED_PTR](#)< [Event](#) > [Ptr](#)
Pointer to a [Event](#) object.

Public Member Functions

- virtual bool [Init](#) ()
Initializes the object deactivated and with automatic reset.
- bool [Init](#) (bool [manual_reset](#), bool [initial_state](#)=false)
Initializes the object.
- virtual [WaitResult](#) [Wait](#) (int [time_out](#)=-1)
Performs a wait operation with the object to get it.
- virtual bool [Dispose](#) ()
*Release the resources associated to the IPC object and sets the internal status to *false*.*
- bool [Set](#) (bool [new_state](#)=true)
Sets the state of the object.

- bool [Get](#) () const
Returns the current activation state of the object.
- bool [Pulse](#) ()
Generates the same result as if the event has automatic reset and the `Set` method is called with `true`, independently of the real reset type.
- bool [Reset](#) ()
Desactivates the object.

Private Attributes

- bool [state](#)
Current activation state of the event.
- bool [manual_reset](#)
Indicates if the event reset is manual.
- pthread_cond_t [condv](#)
Conditional variable information.
- pthread_mutex_t [mutex](#)
Mutex information.

6.24.1 Detailed Description

IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API.

See also

[IPCObject](#)

6.24.2 Member Typedef Documentation

6.24.2.1 typedef SHARED_PTR<Event> ipc::Event::Ptr

Pointer to a [Event](#) object.

6.24.3 Member Function Documentation

6.24.3.1 bool ipc::Event::Dispose () [virtual]

Release the resources associated to the IPC object and sets the internal status to `false`.

Returns

`true` if successful.

Reimplemented from [ipc::IPCObject](#).

Here is the call graph for this function:

6.24.3.2 bool ipc::Event::Get () const [inline]

Returns the current activation state of the object.

6.24.3.3 `virtual bool ipc::Event::Init () [inline],[virtual]`

Initializes the object deactivated and with automatic reset.

Returns

`true` if successful.

Reimplemented from [ipc::IPCObject](#).

6.24.3.4 `bool ipc::Event::Init (bool manual_reset, bool initial_state = false)`

Initializes the object.

Parameters

<i>manual_reset</i>	<code>true</code> if the reset is manual.
<i>initial_state</i>	<code>true</code> if the initial state is activated.

Returns

`true` if successful.

Here is the call graph for this function:

6.24.3.5 `bool ipc::Event::Pulse ()`

Generates the same result as if the event has automatic reset and the `Set` method is called with `true`, independently of the real reset type.

Returns

`true` if successful.

Here is the call graph for this function:

6.24.3.6 `bool ipc::Event::Reset () [inline]`

Deactivates the object.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.24.3.7 `bool ipc::Event::Set (bool new_state = true)`

Sets the state of the object.

If it is activated (with `true`) and the reset is manual, all the threads waiting for the object will be resumed. If the reset is not manual (automatic), only one thread will be resumed and the state will be set to `false` again.

Parameters

<i>new_state</i>	New state of the object.
------------------	--------------------------

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.24.3.8 WaitResult ipc::Event::Wait (int *time_out* = -1) [virtual]

Performs a wait operation with the object to get it.

Parameters

<i>time_out</i>	Time out (infinite by default).
-----------------	---------------------------------

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error.

Reimplemented from [ipc::IPCObject](#).

Here is the call graph for this function:

6.24.4 Member Data Documentation

6.24.4.1 pthread_cond_t ipc::Event::condv [private]

Conditional variable information.

6.24.4.2 bool ipc::Event::manual_reset [private]

Indicates if the event reset is manual.

6.24.4.3 pthread_mutex_t ipc::Event::mutex [private]

[Mutex](#) information.

6.24.4.4 bool ipc::Event::state [private]

Current activation state of the event.

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

- [ipc/event.h](#)
- [ipc/event.cc](#)

6.25 jpeg2000::FileManager Class Reference

Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.

```
#include <file_manager.h>
```

Collaboration diagram for jpeg2000::FileManager:

Public Member Functions

- string [GetCacheFileName](#) (const string &path_image_file)
Returns the cache file name equivalent to the given image file name.
- [FileManager](#) ()
Initializes the object.
- [FileManager](#) (string root_dir, string cache_dir)
Initializes the object.
- bool [Init](#) (string root_dir=".", string cache_dir=".")
Initializes the object.
- string [root_dir](#) () const
Returns the root directory of the image repository.
- string [cache_dir](#) () const
Returns the directory used for caching.
- bool [ReadImage](#) (const string &name_image_file, [ImageInfo](#) *image_info)
Reads an image file and creates the associated cache file if it does not exist yet.
- virtual [~FileManager](#) ()

Private Member Functions

- bool [ExistCacheImage](#) (const string &path_image_file, string *path_cache_file)
Returns true if the cache file exists and it is updated.
- bool [ReadBoxHeader](#) (const [File](#) &file, uint32_t *type_box, uint64_t *length_box)
Reads the header information.
- bool [ReadCodestream](#) (const [File](#) &file, [CodingParameters](#) *params, [CodestreamIndex](#) *index)
Reads the information of a codestream.
- bool [ReadSIZMarker](#) (const [File](#) &file, [CodingParameters](#) *params)
Reads the information of a SIZ marker.
- bool [ReadCODMarker](#) (const [File](#) &file, [CodingParameters](#) *params)
Reads the information of a COD marker.
- bool [ReadSOTMarker](#) (const [File](#) &file, [CodestreamIndex](#) *index)
Reads the information of a SOT marker.
- bool [ReadPLTMarker](#) (const [File](#) &file, [CodestreamIndex](#) *index)
Reads the information of a PLT marker.
- bool [ReadSODMarker](#) (const [File](#) &file, [CodestreamIndex](#) *index)
Reads the information of a SOD marker.
- bool [ReadJP2](#) (const [File](#) &file, [ImageInfo](#) *image_info)
Reads the information of a JP2 image file.
- bool [ReadJPX](#) (const [File](#) &file, [ImageInfo](#) *image_info)
Reads the information of a JPX image file.
- bool [ReadNlStBox](#) (const [File](#) &file, int *num_codestream, int length_box)
Reads the information of a NLST box.
- bool [ReadFlStBox](#) (const [File](#) &file, uint64_t length_box, uint16_t *data_reference)
Reads the information of a FLST box.
- bool [ReadUrlBox](#) (const [File](#) &file, uint64_t length_box, string *path_file)
Reads the information of a URL box.

Private Attributes

- string `root_dir_`
Root directory of the repository.
- string `cache_dir_`
Caching directory.

6.25.1 Detailed Description

Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 jpeg2000::FileManager::FileManager () `[inline]`

Initializes the object.

6.25.2.2 jpeg2000::FileManager::FileManager (string *root_dir*, string *cache_dir*) `[inline]`

Initializes the object.

Parameters

<i>root_dir</i>	Root directory of the image repository.
<i>cache_dir</i>	Directory for caching.

Here is the call graph for this function:

6.25.2.3 virtual jpeg2000::FileManager::~~FileManager () `[inline]`, `[virtual]`

6.25.3 Member Function Documentation

6.25.3.1 string jpeg2000::FileManager::cache_dir () const `[inline]`

Returns the directory used for caching.

Here is the caller graph for this function:

6.25.3.2 bool jpeg2000::FileManager::ExistCacheImage (const string & *path_image_file*, string * *path_cache_file*) `[private]`

Returns `true` if the cache file exists and it is updated.

Parameters

<i>path_image_file</i>	Path of the image file.
<i>path_cache_file</i>	Receives the path of the associated cache file.

6.25.3.3 string jpeg2000::FileManager::GetCacheFileName (const string & *path_image_file*)

Returns the cache file name equivalent to the given image file name.

Here is the caller graph for this function:

6.25.3.4 `bool jpeg2000::FileManager::Init (string root_dir = " ./ ", string cache_dir = " ./ ") [inline]`

Initializes the object.

Parameters

<i>root_dir</i>	Root directory of the image repository.
<i>cache_dir</i>	Directory for caching.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.25.3.5 `bool jpeg2000::FileManager::ReadBoxHeader (const File & fim, uint32_t * type_box, uint64_t * length_box)`
`[private]`

Reads the header information.

of a JP2/JPX box.

Parameters

<i>fim</i>	Image file.
<i>type_box</i>	Receives the type of the box.
<i>length_box</i>	Receives the length of the box.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.6 `bool jpeg2000::FileManager::ReadCodestream (const File & file, CodingParameters * params, CodestreamIndex * index)`
`[private]`

Reads the information of a codestream.

Parameters

<i>file</i>	Image file.
<i>params</i>	Receives the coding parameters.
<i>index</i>	Receives the indexing information.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.7 `bool jpeg2000::FileManager::ReadCODMarker (const File & file, CodingParameters * params)`
`[private]`

Reads the information of a COD marker.

Parameters

<i>file</i>	Image file.
<i>params</i>	Pointer to the coding parameters to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.8 `bool jpeg2000::FileManager::ReadFistBox (const File & file, uint64_t length_box, uint16_t * data_reference)`
`[private]`

Reads the information of a FLST box.

Parameters

<i>file</i>	Image file.
<i>length_box</i>	Box length in bytes.
<i>data_reference</i>	Receives the data reference.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.9 `bool jpeg2000::FileManager::ReadImage (const string & name_image_file, ImageInfo * image_info)`

Reads an image file and creates the associated cache file if it does not exist yet.

Parameters

<i>name_image_↔ file</i>	File name of the image.
<i>image_info</i>	Receives the information of the image.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.10 `bool jpeg2000::FileManager::ReadJP2 (const File & file, ImageInfo * image_info)` `[private]`

Reads the information of a JP2 image file.

Parameters

<i>file</i>	Image file.
<i>image_info</i>	Receives the image information.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.11 `bool jpeg2000::FileManager::ReadJPX (const File & file, ImageInfo * image_info)` `[private]`

Reads the information of a JPX image file.

Parameters

<i>file</i>	Image file.
<i>image_info</i>	Receives the image information.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.12 `bool jpeg2000::FileManager::ReadNlStBox (const File & file, int * num_codestream, int length_box)`
`[private]`

Reads the information of a NLST box.

Parameters

<i>file</i>	Image file.
<i>num_↔ codestream</i>	Receives the number of codestream read.
<i>length_box</i>	Box length in bytes.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.13 `bool jpeg2000::FileManager::ReadPLTMarker (const File & file, CodestreamIndex * index)` `[private]`

Reads the information of a PLT marker.

Parameters

<i>file</i>	Image file.
<i>index</i>	Pointer to the indexing information to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.14 `bool jpeg2000::FileManager::ReadSIZMarker (const File & file, CodingParameters * params)` `[private]`

Reads the information of a SIZ marker.

Parameters

<i>file</i>	Image file.
<i>params</i>	Pointer to the coding parameters to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.15 `bool jpeg2000::FileManager::ReadSODMarker (const File & file, CodestreamIndex * index)` `[private]`

Reads the information of a SOD marker.

Parameters

<i>file</i>	Image file.
<i>index</i>	Pointer to the indexing information to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.16 `bool jpeg2000::FileManager::ReadSOTMarker (const File & file, CodestreamIndex * index)` `[private]`

Reads the information of a SOT marker.

Parameters

<i>file</i>	Image file.
<i>index</i>	Pointer to the indexing information to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.17 `bool jpeg2000::FileManager::ReadUrlBox (const File & file, uint64_t length_box, string * path_file)`
`[private]`

Reads the information of a URL box.

Parameters

<i>file</i>	Image file.
<i>length_box</i>	Box length in bytes.
<i>path_file</i>	Receives the URL path read.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.18 `string jpeg2000::FileManager::root_dir () const` `[inline]`

Returns the root directory of the image repository.

Here is the caller graph for this function:

6.25.4 Member Data Documentation

6.25.4.1 `string jpeg2000::FileManager::cache_dir_` `[private]`

Caching directory.

6.25.4.2 string jpeg2000::FileManager::root_dir_ [private]

Root directory of the repository.

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

- [jpeg2000/file_manager.h](#)
- [jpeg2000/file_manager.cc](#)

6.26 data::FileSegment Class Reference

Identifies a data segment of a file.

```
#include <file_segment.h>
```

Collaboration diagram for data::FileSegment:

Public Member Functions

- [FileSegment](#) ()
Initializes all the member variables with zero, being a null segment.
- [FileSegment](#) (uint64_t offset, uint64_t length)
Initializes the segment with the given parameters.
- [FileSegment](#) (const [FileSegment](#) &segment)
Copy constructor.
- [FileSegment](#) & operator= (const [FileSegment](#) &segment)
Copy assignment.
- [FileSegment](#) & RemoveFirst (int count)
Removes the first bytes of the segment.
- [FileSegment](#) & RemoveLast (int count)
Removes the last bytes of the segment.
- bool IsContiguousTo (const [FileSegment](#) &segment) const
Returns true if the segment is contiguous to another given segment, so the first byte of the given segment is just the next byte after the last byte of the segment.
- bool operator== (const [FileSegment](#) &segment) const
- bool operator!= (const [FileSegment](#) &segment) const
- template<typename T >
T & SerializeWith (T &stream)
- virtual ~[FileSegment](#) ()

Public Attributes

- uint64_t offset
Offset of the data segment.
- uint64_t length
Length of the data segment.

Static Public Attributes

- static const [FileSegment](#) Null
Identifies a null segment, with the offset as well as the length set to zero.

Friends

- ostream & [operator<<](#) (ostream &out, const [FileSegment](#) &segment)

6.26.1 Detailed Description

Identifies a data segment of a file.

This segment is defined by an offset and a length (number of bytes), both of them with an unsigned integer of 64 bits. This class is serializable and can be printed.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 data::FileSegment::FileSegment () [\[inline\]](#)

Initializes all the member variables with zero, being a null segment.

6.26.2.2 data::FileSegment::FileSegment (uint64_t *offset*, uint64_t *length*) [\[inline\]](#)

Initializes the segment with the given parameters.

Parameters

<i>offset</i>	Offset of the segment.
<i>length</i>	Length of the segment.

6.26.2.3 data::FileSegment::FileSegment (const [FileSegment](#) & *segment*) [\[inline\]](#)

Copy constructor.

6.26.2.4 virtual data::FileSegment::~~FileSegment () [\[inline\]](#), [\[virtual\]](#)

6.26.3 Member Function Documentation

6.26.3.1 bool data::FileSegment::IsContiguousTo (const [FileSegment](#) & *segment*) const [\[inline\]](#)

Returns `true` if the segment is contiguous to another given segment, so the first byte of the given segment is just the next byte after the last byte of the segment.

6.26.3.2 bool data::FileSegment::operator!= (const [FileSegment](#) & *segment*) const [\[inline\]](#)

6.26.3.3 [FileSegment](#)& data::FileSegment::operator= (const [FileSegment](#) & *segment*) [\[inline\]](#)

Copy assignment.

6.26.3.4 bool data::FileSegment::operator== (const [FileSegment](#) & *segment*) const [\[inline\]](#)

6.26.3.5 [FileSegment](#)& data::FileSegment::RemoveFirst (int *count*) [\[inline\]](#)

Removes the first bytes of the segment.

Modifies the segment as if a number of bytes (specified by the parameter) was removed from the beginning of the segment.

Parameters

<i>count</i>	Number of bytes to remove.
--------------	----------------------------

Returns

**this.*

Here is the caller graph for this function:

6.26.3.6 FileSegment& data::FileSegment::RemoveLast (int *count*) [inline]

Removes the last bytes of the segment.

Modifies the segment as if a number of bytes (specified by the parameter) was removed from the end of the segment.

Parameters

<i>count</i>	Number of bytes to remove.
--------------	----------------------------

Returns

**this.*

6.26.3.7 template<typename T > T& data::FileSegment::SerializeWith (T & *stream*) [inline]

6.26.4 Friends And Related Function Documentation

6.26.4.1 ostream& operator<< (ostream & *out*, const FileSegment & *segment*) [friend]

6.26.5 Member Data Documentation

6.26.5.1 uint64_t data::FileSegment::length

Length of the data segment.

6.26.5.2 const FileSegment data::FileSegment::Null [static]

Identifies a null segment, with the offset as well as the length set to zero.

6.26.5.3 uint64_t data::FileSegment::offset

Offset of the data segment.

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

- [data/file_segment.h](#)
- [data/file_segment.cc](#)

6.27 http::Header Class Reference

Class used to handle a HTTP header.

```
#include <header.h>
```

Inheritance diagram for http::Header:

Collaboration diagram for http::Header:

Public Types

- typedef [HeaderBase](#)< [HeaderName::CONTENT_TYPE](#) > [ContentType](#)
Predefined "Content-Type".
- typedef [HeaderBase](#)< [HeaderName::CACHE_CONTROL](#) > [CacheControl](#)
Predefined "Cache-Control" header.
- typedef [HeaderBase](#)< [HeaderName::CONTENT_LENGTH](#) > [ContentLength](#)
Predefined "Content-Length" header.
- typedef [HeaderBase](#)< [HeaderName::TRANSFER_ENCODING](#) > [TransferEncoding](#)
Predefined "Transfer-Encoding" header.

Public Member Functions

- [Header](#) ()
Empty constructor.
- [Header](#) (const string &[name](#), const string &[value](#))
Initializes the header content (name and value).

Friends

- template<const char * NAME>
bool [operator==](#) (const [Header](#) &a, const [HeaderBase](#)< NAME > &b)
Returns `true` if the names of the two headers are equal.

Additional Inherited Members

6.27.1 Detailed Description

Class used to handle a HTTP header.

See also

[HeaderBase](#)
[HeaderName](#)

6.27.2 Member Typedef Documentation

6.27.2.1 typedef [HeaderBase](#)<[HeaderName::CACHE_CONTROL](#)> [http::Header::CacheControl](#)

Predefined "Cache-Control" header.

6.27.2.2 typedef [HeaderBase](#)<[HeaderName::CONTENT_LENGTH](#)> [http::Header::ContentLength](#)

Predefined "Content-Length" header.

6.27.2.3 typedef [HeaderBase](#)<[HeaderName::CONTENT_TYPE](#)> [http::Header::ContentType](#)

Predefined "Content-Type".

6.27.2.4 typedef [HeaderBase](#)<[HeaderName::TRANSFER_ENCODING](#)> [http::Header::TransferEncoding](#)

Predefined "Transfer-Encoding" header.

6.27.3 Constructor & Destructor Documentation

6.27.3.1 `http::Header::Header () [inline]`

Empty constructor.

6.27.3.2 `http::Header::Header (const string & name, const string & value) [inline]`

Initializes the header content (name and value).

Parameters

<i>name</i>	Header name.
<i>value</i>	Header value.

6.27.4 Friends And Related Function Documentation

6.27.4.1 `template<const char * NAME> bool operator== (const Header & a, const HeaderBase< NAME > & b) [friend]`

Returns `true` if the names of the two headers are equal.

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

- [http/header.h](#)

6.28 `http::HeaderBase< NAME >` Class Template Reference

Template class used to identify a HTTP header.

```
#include <header.h>
```

Collaboration diagram for `http::HeaderBase< NAME >`:

Public Member Functions

- [HeaderBase](#) ()
Empty constructor.
- [HeaderBase](#) (const string &[value](#))
Initializes the header value.

Static Public Member Functions

- static const char * [name](#) ()
Returns the name of the header, used in the specialization of the class.

Private Attributes

- string [value](#)
String value of the header.

Friends

- ostream & [operator<<](#) (ostream &out, const [HeaderBase](#) &header)
- istream & [operator>>](#) (istream &in, [HeaderBase](#) &header)

6.28.1 Detailed Description

template<const char * NAME>class http::HeaderBase< NAME >

Template class used to identify a HTTP header.

It is possible to use this class with standard streams. This class is specialized with the header name.

See also

[Header](#)

6.28.2 Constructor & Destructor Documentation

6.28.2.1 template<const char * NAME> http::HeaderBase< NAME >::HeaderBase () [inline]

Empty constructor.

6.28.2.2 template<const char * NAME> http::HeaderBase< NAME >::HeaderBase (const string & value) [inline]

Initializes the header value.

6.28.3 Member Function Documentation

6.28.3.1 template<const char * NAME> static const char* http::HeaderBase< NAME >::name () [inline], [static]

Returns the name of the header, used in the specialization of the class.

Here is the caller graph for this function:

6.28.4 Friends And Related Function Documentation

6.28.4.1 template<const char * NAME> ostream& operator<< (ostream & out, const HeaderBase< NAME > & header) [friend]

6.28.4.2 template<const char * NAME> istream& operator>> (istream & in, HeaderBase< NAME > & header) [friend]

6.28.5 Member Data Documentation

6.28.5.1 template<const char * NAME> string http::HeaderBase< NAME >::value [private]

String value of the header.

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

- [http/header.h](#)

6.29 http::HeaderBase< HeaderName::UNDEFINED > Class Template Reference

Specialization of the [HeaderBase](#) template class with the [HeaderName::UNDEFINED](#) value.

```
#include <header.h>
```

Inheritance diagram for `http::HeaderBase< HeaderName::UNDEFINED >`:

Collaboration diagram for `http::HeaderBase< HeaderName::UNDEFINED >`:

Public Member Functions

- [HeaderBase](#) ()
Empty constructor.
- [HeaderBase](#) (const string &[name](#), const string &[value](#))
Initializes the header content (name and value).

Public Attributes

- string [name](#)
Header name.
- string [value](#)
Header value.

Friends

- ostream & [operator<<](#) (ostream &out, const [HeaderBase](#) &header)
- istream & [operator>>](#) (istream &in, [HeaderBase](#) &header)

6.29.1 Detailed Description

```
template<>class http::HeaderBase< HeaderName::UNDEFINED >
```

Specialization of the [HeaderBase](#) template class with the [HeaderName::UNDEFINED](#) value.

In this case the header name is not fixed, handled by an internal variable. This class is used as base for the class [Header](#).

See also

[Header](#)

6.29.2 Constructor & Destructor Documentation

6.29.2.1 `http::HeaderBase< HeaderName::UNDEFINED >::HeaderBase ()` `[inline]`

Empty constructor.

6.29.2.2 `http::HeaderBase< HeaderName::UNDEFINED >::HeaderBase (const string & name, const string & value)` `[inline]`

Initializes the header content (name and value).

Parameters

<i>name</i>	Header name.
<i>value</i>	Header value.

Here is the call graph for this function:

6.29.3 Friends And Related Function Documentation

6.29.3.1 `ostream& operator<< (ostream & out, const HeaderBase< HeaderName::UNDEFINED > & header)`
[friend]

6.29.3.2 `istream& operator>> (istream & in, HeaderBase< HeaderName::UNDEFINED > & header)` [friend]

6.29.4 Member Data Documentation

6.29.4.1 `string http::HeaderBase< HeaderName::UNDEFINED >::name`

[Header](#) name.

6.29.4.2 `string http::HeaderBase< HeaderName::UNDEFINED >::value`

[Header](#) value.

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

- [http/header.h](http://header.h)

6.30 http::HeaderName Class Reference

Container for the strings associated to the most common HTTP headers, used for the specialization of the class [HeaderBase](#).

```
#include <header.h>
```

Collaboration diagram for http::HeaderName:

Static Public Attributes

- static const char [UNDEFINED](#) [] = ""
No header name defined.
- static const char [CONTENT_TYPE](#) [] = "Content-Type"
The header Content-Type
- static const char [CACHE_CONTROL](#) [] = "Cache-Control"
The header Cache-Control
- static const char [CONTENT_LENGTH](#) [] = "Content-Length"
The header Content-Length
- static const char [TRANSFER_ENCODING](#) [] = "Transfer-Encoding"
The header Transfer-Encoding

6.30.1 Detailed Description

Container for the strings associated to the most common HTTP headers, used for the specialization of the class [HeaderBase](#).

See also

[HeaderBase](#)

6.30.2 Member Data Documentation

6.30.2.1 `const char http::HeaderName::CACHE_CONTROL = "Cache-Control" [static]`

The header `Cache-Control`

6.30.2.2 `const char http::HeaderName::CONTENT_LENGTH = "Content-Length" [static]`

The header `Content-Length`

6.30.2.3 `const char http::HeaderName::CONTENT_TYPE = "Content-Type" [static]`

The header `Content-Type`

6.30.2.4 `const char http::HeaderName::TRANSFER_ENCODING = "Transfer-Encoding" [static]`

The header `Transfer-Encoding`

6.30.2.5 `const char http::HeaderName::UNDEFINED = "" [static]`

No header name defined.

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

- [http/header.h](#)
- [http/header.cc](#)

6.31 jpeg2000::ImageIndex Class Reference

Contains the indexing information of a JPEG2000 image file that is managed by the index manager.

```
#include <image_index.h>
```

Collaboration diagram for jpeg2000::ImageIndex:

Public Types

- typedef list< [ImageIndex](#) >::iterator [Ptr](#)
Pointer of an object of this class.

Public Member Functions

- [ImageIndex](#) (const [ImageIndex](#) &image_index)
Copy constructor.
- int [GetNumCodestreams](#) () const
Returns the number of codestreams.
- int [GetNumMetadatas](#) () const
Returns the number of meta-data blocks.

- bool [ReadLock](#) (const [Range](#) &range=[Range](#)(0, 0))
Gets the lock for reading, for a specific range of codestreams.
- bool [ReadUnlock](#) (const [Range](#) &range=[Range](#)(0, 0))
Releases the lock for reading, for a specific range of codestreams.
- string [GetPathName](#) () const
Returns the path name of the image.
- string [GetPathName](#) (int num_codestream) const
Returns the path name of a given codestream, if it is a hyperlinked codestream.
- [FileSegment](#) [GetMainHeader](#) (int num_codestream) const
Returns the file segment the main header of a given codestream.
- [FileSegment](#) [GetMetadata](#) (int num_metadata) const
Returns the file segment of a meta-data block.
- [Placeholder](#) [GetPlaceholder](#) (int num_placeholder) const
Returns the information of a place-holder.
- [FileSegment](#) [GetPacket](#) (int num_codestream, const [Packet](#) &packet, int *offset=NULL)
Returns the file segment of a packet.
- [CodingParameters::Ptr](#) [GetCodingParameters](#) () const
Returns a pointer to the coding parameters.
- bool [IsHyperLinked](#) (int num_codestream) const
Returns `true` if the image contains hyperlinks.
- [Ptr](#) [GetHyperLink](#) (int num_codestream) const
Returns a pointer to a hyperlink.
- int [GetNumHyperLinks](#) () const
Returns the number of hyperlinks.
- [operator CodingParameters::Ptr](#) () const
- [ImageIndex](#) & [operator=](#) (const [ImageIndex](#) &image_index)
- virtual [~ImageIndex](#) ()

Private Member Functions

- bool [GetPLTLength](#) (const [File](#) &file, int ind_codestream, uint64_t *length_packet)
Gets the packet lengths from a PLT marker.
- void [GetOffsetPacket](#) (const [File](#) &file, int ind_codestream, uint64_t length_packet)
Gets the packet offsets.
- bool [BuildIndex](#) (int ind_codestream, int max_index)
Builds the required index for the required resolution levels.
- bool [Init](#) (const string &path_name, const [ImageInfo](#) &image_info)
Initializes the object.
- bool [Init](#) (const string &path_name, [CodingParameters::Ptr](#) coding_parameters, const [ImageInfo](#) &image_info, int index)
Initializes the object.
- [ImageIndex](#) ()
Empty constructor.

Private Attributes

- [RdWrLock::Ptr rdwr_lock](#)
Read/write lock.
- `vector< int >` [last_plt](#)
- `vector< int >` [last_packet](#)
- `vector< uint64_t >` [last_offset_PLT](#)
- `vector< uint64_t >` [last_offset_packet](#)
- `string` [path_name](#)
Image file name.
- [Metadata meta_data](#)
Image Metadata.
- `int` [num_references](#)
Number of references.
- `vector< int >` [max_resolution](#)
Maximum resolution number.
- `vector< PacketIndex >` [packet_indexes](#)
Code-stream packet index.
- `vector< CodestreamIndex >` [codestreams](#)
Image code-streams.
- [CodingParameters::Ptr](#) [coding_parameters](#)
Image coding parameters.
- `vector< list< ImageIndex >::iterator >` [hyper_links](#)
Image hyperlinks.

Friends

- class [IndexManager](#)
- `ostream &` [operator<<](#) (`ostream &out`, `const ImageIndex &info_node`)

6.31.1 Detailed Description

Contains the indexing information of a JPEG2000 image file that is managed by the index manager.

This class can be printed.

Maintains a read/write lock for controlling the multi-thread access to the indexing information. For instance, by default all the threads usually want to read the information. The packet index built on demand, so only when a thread wants to create a new level of the packet index, it needs to write.

See also

[IndexManager](#)

6.31.2 Member Typedef Documentation

6.31.2.1 `typedef list<ImageIndex>::iterator jpeg2000::ImageIndex::Ptr`

Pointer of an object of this class.

6.31.3 Constructor & Destructor Documentation

6.31.3.1 jpeg2000::ImageIndex::ImageIndex () [inline], [private]

Empty constructor.

Only the index manager can use this constructor.

6.31.3.2 jpeg2000::ImageIndex::ImageIndex (const ImageIndex & *image_index*) [inline]

Copy constructor.

6.31.3.3 virtual jpeg2000::ImageIndex::~~ImageIndex () [inline], [virtual]

6.31.4 Member Function Documentation

6.31.4.1 bool jpeg2000::ImageIndex::BuildIndex (int *ind_codestream*, int *max_index*) [private]

Builds the required index for the required resolution levels.

Parameters

<i>ind_codestream</i>	Codestream index.
<i>max_index</i>	Maximum resolution level.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.31.4.2 CodingParameters::Ptr jpeg2000::ImageIndex::GetCodingParameters () const [inline]

Returns a pointer to the coding parameters.

6.31.4.3 Ptr jpeg2000::ImageIndex::GetHyperLink (int *num_codestream*) const [inline]

Returns a pointer to a hyperlink.

Parameters

<i>num_↔ codestream</i>	Number of the hyperlink (codestream).
-----------------------------	---------------------------------------

6.31.4.4 FileSegment jpeg2000::ImageIndex::GetMainHeader (int *num_codestream*) const [inline]

Returns the file segment the main header of a given codestream.

Parameters

<i>num_↔ codestream</i>	Codestream number
-----------------------------	-------------------

6.31.4.5 **FileSegment** jpeg2000::ImageIndex::GetMetadata (int *num_metadata*) const [inline]

Returns the file segment of a meta-data block.

Parameters

<i>num_metadata</i>	Meta-data number.
---------------------	-------------------

6.31.4.6 `int jpeg2000::ImageIndex::GetNumCodestreams () const [inline]`

Returns the number of codestreams.

6.31.4.7 `int jpeg2000::ImageIndex::GetNumHyperLinks () const [inline]`

Returns the number of hyperlinks.

6.31.4.8 `int jpeg2000::ImageIndex::GetNumMetadatas () const [inline]`

Returns the number of meta-data blocks.

6.31.4.9 `void jpeg2000::ImageIndex::GetOffsetPacket (const File & file, int ind_codestream, uint64_t length_packet) [private]`

Gets the packet offsets.

Parameters

<i>file</i>	File where to read the data from.
<i>ind_codestream</i>	Codestream index.
<i>length_packet</i>	Packet length.

Returns

`true` if successful.

Here is the caller graph for this function:

6.31.4.10 `FileSegment jpeg2000::ImageIndex::GetPacket (int num_codestream, const Packet & packet, int * offset = NULL)`

Returns the file segment of a packet.

Parameters

<i>num_↔ codestream</i>	Codestream number.
<i>packet</i>	Packet information.
<i>offset</i>	If it is not <code>NULL</code> receives the offset of the packet.

Here is the call graph for this function:

6.31.4.11 `string jpeg2000::ImageIndex::GetPathName () const [inline]`

Returns the path name of the image.

6.31.4.12 `string jpeg2000::ImageIndex::GetPathName (int num_codestream) const [inline]`

Returns the path name of a given codestream, if it is a hyperlinked codestream.

Parameters

<i>num_↔ codestream</i>	Codestream number.
-----------------------------	--------------------

6.31.4.13 Placeholder jpeg2000::ImageIndex::GetPlaceholder (int *num_placeholder*) const [inline]

Returns the information of a place-holder.

Parameters

<i>num_↔ placeholder</i>	Place-holder number.
------------------------------	----------------------

6.31.4.14 bool jpeg2000::ImageIndex::GetPLTLength (const File & *file*, int *ind_codestream*, uint64_t * *length_packet*) [private]

Gets the packet lengths from a PLT marker.

Parameters

<i>file</i>	File where to read the data from.
<i>ind_codestream</i>	Codestream index.
<i>length_packet</i>	It is returned the length of the packet.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.31.4.15 bool jpeg2000::ImageIndex::Init (const string & *path_name*, const ImageInfo & *image_info*) [private]

Initializes the object.

Parameters

<i>path_name</i>	Path name of the image.
<i>image_info</i>	Indexing image information.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.31.4.16 bool jpeg2000::ImageIndex::Init (const string & *path_name*, CodingParameters::Ptr *coding_parameters*, const ImageInfo & *image_info*, int *index*) [private]

Initializes the object.

Parameters

<i>path_name</i>	Path name of the image.
<i>coding_↔ parameters</i>	Coding parameters.
<i>image_info</i>	Indexing image information.
<i>index</i>	Image index.

Returns

`true` if successful

6.31.4.17 `bool jpeg2000::ImageIndex::IsHyperLinked (int num_codestream) const [inline]`

Returns `true` if the image contains hyperlinks.

6.31.4.18 `jpeg2000::ImageIndex::operator CodingParameters::Ptr () const [inline]`

6.31.4.19 `ImageIndex& jpeg2000::ImageIndex::operator= (const ImageIndex & image_index) [inline]`

Here is the call graph for this function:

6.31.4.20 `bool jpeg2000::ImageIndex::ReadLock (const Range & range = Range (0, 0))`

Gets the lock for reading, for a specific range of codestreams.

Returns

`true` if successful

6.31.4.21 `bool jpeg2000::ImageIndex::ReadUnlock (const Range & range = Range (0, 0))`

Releases the lock for reading, for a specific range of codestreams.

Returns

`true` if successful

6.31.5 Friends And Related Function Documentation

6.31.5.1 `friend class IndexManager [friend]`

6.31.5.2 `ostream& operator<< (ostream & out, const ImageIndex & info_node) [friend]`

6.31.6 Member Data Documentation

6.31.6.1 `vector<CodestreamIndex> jpeg2000::ImageIndex::codestreams [private]`

Image code-streams.

6.31.6.2 `CodingParameters::Ptr jpeg2000::ImageIndex::coding_parameters [private]`

Image coding parameters.

6.31.6.3 `vector<list<ImageIndex>::iterator> jpeg2000::ImageIndex::hyper_links` [private]

Image hyperlinks.

6.31.6.4 `vector<uint64_t> jpeg2000::ImageIndex::last_offset_packet` [private]

6.31.6.5 `vector<uint64_t> jpeg2000::ImageIndex::last_offset_PLT` [private]

6.31.6.6 `vector<int> jpeg2000::ImageIndex::last_packet` [private]

6.31.6.7 `vector<int> jpeg2000::ImageIndex::last_plt` [private]

6.31.6.8 `vector<int> jpeg2000::ImageIndex::max_resolution` [private]

Maximum resolution number.

6.31.6.9 **Metadata** `jpeg2000::ImageIndex::meta_data` [private]

Image [Metadata](#).

6.31.6.10 `int jpeg2000::ImageIndex::num_references` [private]

Number of references.

6.31.6.11 `vector<PacketIndex> jpeg2000::ImageIndex::packet_indexes` [private]

Code-stream packet index.

6.31.6.12 `string jpeg2000::ImageIndex::path_name` [private]

Image file name.

6.31.6.13 **RdWrLock::Ptr** `jpeg2000::ImageIndex::rdwr_lock` [private]

Read/write lock.

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

- [jpeg2000/image_index.h](#)
- [jpeg2000/image_index.cc](#)

6.32 jpeg2000::ImageInfo Class Reference

Contains the indexing information of a JPEG2000 image.

```
#include <image_info.h>
```

Collaboration diagram for jpeg2000::ImageInfo:

Public Member Functions

- [ImageInfo](#) ()
Empty constructor.
- [ImageInfo](#) (const [ImageInfo](#) &info)
Copy constructor.
- const [ImageInfo](#) & [operator=](#) (const [ImageInfo](#) &info)
Copy assignment.
- template<typename T >
T & [SerializeWith](#) (T &stream)
- virtual [~ImageInfo](#) ()

Public Attributes

- [Metadata](#) meta_data
Meta-data information.
- multimap< string, int > [paths](#)
Paths of the hyperlinks (if any)
- [CodingParameters](#) coding_parameters
Coding parameters.
- vector< [CodestreamIndex](#) > [codestreams](#)
Codestreams information.
- vector< [Metadata](#) > [meta_data_hyperlinks](#)
Meta-data of the hyperlinks.

Friends

- ostream & [operator<<](#) (ostream &out, const [ImageInfo](#) &info)

6.32.1 Detailed Description

Contains the indexing information of a JPEG2000 image.

This class can be serialized and printed.

See also

[CodingParameters](#)
[CodestreamIndex](#)
[Metadata](#)

6.32.2 Constructor & Destructor Documentation

6.32.2.1 jpeg2000::ImageInfo::ImageInfo () [\[inline\]](#)

Empty constructor.

6.32.2.2 jpeg2000::ImageInfo::ImageInfo (const ImageInfo & info) [\[inline\]](#)

Copy constructor.

6.32.2.3 `virtual jpeg2000::ImageInfo::~ImageInfo () [inline],[virtual]`

6.32.3 Member Function Documentation

6.32.3.1 `const ImageInfo& jpeg2000::ImageInfo::operator= (const ImageInfo & info) [inline]`

Copy assignment.

Here is the call graph for this function:

6.32.3.2 `template<typename T> T& jpeg2000::ImageInfo::SerializeWith (T & stream) [inline]`

6.32.4 Friends And Related Function Documentation

6.32.4.1 `ostream& operator<< (ostream & out, const ImageInfo & info) [friend]`

6.32.5 Member Data Documentation

6.32.5.1 `vector<CodestreamIndex> jpeg2000::ImageInfo::codestreams`

Codestreams information.

6.32.5.2 `CodingParameters jpeg2000::ImageInfo::coding_parameters`

Coding parameters.

6.32.5.3 `Metadata jpeg2000::ImageInfo::meta_data`

Meta-data information.

6.32.5.4 `vector<Metadata> jpeg2000::ImageInfo::meta_data_hyperlinks`

Meta-data of the hyperlinks.

6.32.5.5 `multimap<string, int> jpeg2000::ImageInfo::paths`

Paths of the hyperlinks (if any)

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

- [jpeg2000/image_info.h](#)

6.33 jpeg2000::IndexManager Class Reference

Manages the indexing information of a repository fo images.

```
#include <index_manager.h>
```

Collaboration diagram for jpeg2000::IndexManager:

Public Member Functions

- [IndexManager](#) ()

- *Empty constructor.*
- [bool Init](#) (string root_dir, string cache_dir)
Initializes the object.
- [ImageIndex::Ptr GetBegin](#) ()
Returns a pointer to the first image index.
- [ImageIndex::Ptr GetEnd](#) ()
Returns a pointer to the last image index.
- [FileManager & file_manager](#) ()
Returns a reference to the base file manager.
- [bool OpenImage](#) (string &path_image_file, [ImageIndex::Ptr](#) *image_index)
Opens an image and adds its index to the list.
- [bool CloseImage](#) (const [ImageIndex::Ptr](#) &image_index)
Closes an image and removes its index from the list, only if it is not used by any other one.
- [int GetSize](#) () const
Returns the size of the list.
- [virtual ~IndexManager](#) ()

Private Member Functions

- [bool UnsafeOpenImage](#) (string &path_image_file, [ImageIndex::Ptr](#) *image_index)
Unsafely (without mutex) opens an image and adds its index to the list.
- [bool UnsafeCloseImage](#) (const [ImageIndex::Ptr](#) &image_index)
Unsafely (without mutex) closes an image and removes its index from the list, only if it is not used by any other one.

Private Attributes

- [Mutex mutex](#)
Mutex for the operations with the list.
- [FileManager file_manager_](#)
File manager.
- [list< ImageIndex > index_list](#)
List of the indexes.

6.33.1 Detailed Description

Manages the indexing information of a repository fo images.

Maintains a list in memory of the indexes (using the class [ImageIndex](#) for the nodes) of all the opened images and allows a multi-thread access to the information.

See also

[FileManager](#)
[ImageIndex](#)

6.33.2 Constructor & Destructor Documentation

6.33.2.1 jpeg2000::IndexManager::IndexManager () [inline]

Empty constructor.

6.33.2.2 `virtual jpeg2000::IndexManager::~~IndexManager () [inline],[virtual]`

6.33.3 Member Function Documentation

6.33.3.1 `bool jpeg2000::IndexManager::CloseImage (const ImageIndex::Ptr & image_index)`

Closes an image and removes its index from the list, only if it is not used by any other one.

Parameters

<i>image_index</i>	Associated image index.
--------------------	-------------------------

Returns

`true` if successful.

Here is the caller graph for this function:

6.33.3.2 `FileManager& jpeg2000::IndexManager::file_manager () [inline]`

Returns a reference to the base file manager.

Here is the caller graph for this function:

6.33.3.3 `ImageIndex::Ptr jpeg2000::IndexManager::GetBegin () [inline]`

Returns a pointer to the first image index.

Here is the caller graph for this function:

6.33.3.4 `ImageIndex::Ptr jpeg2000::IndexManager::GetEnd () [inline]`

Returns a pointer to the last image index.

Here is the caller graph for this function:

6.33.3.5 `int jpeg2000::IndexManager::GetSize () const [inline]`

Returns the size of the list.

Here is the caller graph for this function:

6.33.3.6 `bool jpeg2000::IndexManager::Init (string root_dir, string cache_dir) [inline]`

Initializes the object.

Parameters

<i>root_dir</i>	Root directory of the image repository.
<i>cache_dir</i>	Directory used for caching.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.33.3.7 `bool jpeg2000::IndexManager::OpenImage (string & path_image_file, ImageIndex::Ptr * image_index)`

Opens an image and adds its index to the list.

Parameters

<i>path_image_file</i>	Path of the image file.
<i>image_index</i>	Receives the pointer to the image index created.

Returns

`true` if successful.

Here is the caller graph for this function:

6.33.3.8 `bool jpeg2000::IndexManager::UnsafeCloseImage (const ImageIndex::Ptr & image_index) [private]`

Unsafely (without mutex) closes an image and removes its index from the list, only if it is not used by any other one.

Parameters

<i>image_index</i>	Associated image index.
--------------------	-------------------------

Returns

`true` if successful.

6.33.3.9 `bool jpeg2000::IndexManager::UnsafeOpenImage (string & path_image_file, ImageIndex::Ptr * image_index) [private]`

Unsafely (without mutex) opens an image and adds its index to the list.

Parameters

<i>path_image_file</i>	Path of the image file.
<i>image_index</i>	Receives the pointer to the image index created.

Returns

`true` if successful.

Here is the call graph for this function:

6.33.4 Member Data Documentation

6.33.4.1 `FileManager jpeg2000::IndexManager::file_manager_ [private]`

File manager.

6.33.4.2 `list<ImageIndex> jpeg2000::IndexManager::index_list [private]`

List of the indexes.

6.33.4.3 `Mutex jpeg2000::IndexManager::mutex [private]`

Mutex for the operations with the list.

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

- [jpeg2000/index_manager.h](#)
- [jpeg2000/index_manager.cc](#)

6.34 net::InetAddress Class Reference

Class to identify and handle an Internet address.

```
#include <address.h>
```

Inheritance diagram for net::InetAddress:

Collaboration diagram for net::InetAddress:

Public Member Functions

- [InetAddress](#) ()
Initializes the address to zero.
- [InetAddress](#) (const [InetAddress](#) &address)
Copy constructor.
- [InetAddress](#) (int port)
Initializes the address with given port.
- [InetAddress](#) (const char *path, int port)
Initializes the address with the given path and port.
- [InetAddress](#) & [operator=](#) (const [InetAddress](#) &address)
Copy assignment.
- virtual sockaddr * [GetSockAddr](#) () const
Overloaded from the base class to use the internal address structure.
- virtual int [GetSize](#) () const
Overloaded from the base class to use the internal address structure.
- string [GetPath](#) () const
Returns the address path.
- int [GetPort](#) () const
Returns the port number.

Private Attributes

- sockaddr_in [sock_addr](#)
Internal address structure.

6.34.1 Detailed Description

Class to identify and handle an Internet address.

The used internal address structure is `sockaddr_in`.

See also

[Address](#)

6.34.2 Constructor & Destructor Documentation

6.34.2.1 net::InetAddress::InetAddress () [inline]

Initializes the address to zero.

6.34.2.2 `net::InetAddress::InetAddress (const InetAddress & address) [inline]`

Copy constructor.

6.34.2.3 `net::InetAddress::InetAddress (int port) [inline]`

Initializes the address with given port.

The used path is `INADDR_ANY`.

Parameters

<i>port</i>	Port number.
-------------	--------------

6.34.2.4 `net::InetAddress::InetAddress (const char * path, int port) [inline]`

Initializes the address with the given path and port.

Parameters

<i>path</i>	Address path.
<i>port</i>	Port number.

6.34.3 Member Function Documentation

6.34.3.1 `string net::InetAddress::GetPath () const [inline]`

Returns the address path.

Here is the caller graph for this function:

6.34.3.2 `int net::InetAddress::GetPort () const [inline]`

Returns the port number.

Here is the caller graph for this function:

6.34.3.3 `virtual int net::InetAddress::GetSize () const [inline],[virtual]`

Overloaded from the base class to use the internal address structure.

Implements [net::Address](#).

6.34.3.4 `virtual sockaddr* net::InetAddress::GetSockAddr () const [inline],[virtual]`

Overloaded from the base class to use the internal address structure.

Implements [net::Address](#).

6.34.3.5 `InetAddress& net::InetAddress::operator= (const InetAddress & address) [inline]`

Copy assignment.

6.34.4 Member Data Documentation

6.34.4.1 sockaddr_in net::InetAddress::sock_addr [private]

Internal address structure.

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

- net/[address.h](#)

6.35 data::InputOperator Struct Reference

This struct identifies a basic input operator to be applied to a `File` object.

```
#include <serialize.h>
```

Collaboration diagram for data::InputOperator:

Static Public Member Functions

- static const char * [FileAccess](#) ()
Returns the required file access for this operator.
- static bool [SerializeBytes](#) ([File](#) &file, void *ptr, int num_bytes)
Performs an input (read) serialization of bytes for a file.

6.35.1 Detailed Description

This struct identifies a basic input operator to be applied to a `File` object.

See also

[BaseStream](#)
[File](#)

6.35.2 Member Function Documentation

6.35.2.1 static const char* data::InputOperator::FileAccess () [inline], [static]

Returns the required file access for this operator.

6.35.2.2 static bool data::InputOperator::SerializeBytes ([File](#) &file, void *ptr, int num_bytes) [inline], [static]

Performs an input (read) serialization of bytes for a file.

Parameters

<i>file</i>	File to use for the operation.
<i>ptr</i>	Pointer to the buffer where to store the bytes.
<i>num_bytes</i>	Number of bytes to read from the file.

Returns

`true` if successful.

Here is the call graph for this function:

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

- [data/serialize.h](#)

6.36 data::InputStream Class Reference

Specialization of the [BaseStream](#) for input serializations.

```
#include <serialize.h>
```

Inheritance diagram for data::InputStream:

Collaboration diagram for data::InputStream:

Public Member Functions

- `template<typename T>`
`InputStream & Serialize (T &var)`

Additional Inherited Members

6.36.1 Detailed Description

Specialization of the [BaseStream](#) for input serializations.

See also

[BaseStream](#)

6.36.2 Member Function Documentation

6.36.2.1 `template<typename T> InputStream& data::InputStream::Serialize (T & var)` `[inline]`

Here is the call graph for this function:

Here is the caller graph for this function:

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

- [data/serialize.h](#)

6.37 ipc::IPCObject Class Reference

Class base of all the IPC classes that has the basic operations (`Init`, `Wait` and `Dispose`) to be overloaded.

```
#include <ipc_object.h>
```

Inheritance diagram for ipc::IPCObject:

Collaboration diagram for ipc::IPCObject:

Public Types

- typedef `SHARED_PTR< IPCObject > Ptr`
Pointer to an IPC object.

Public Member Functions

- `IPCObject ()`
Initializes the internal status to `false`.
- virtual `bool Init ()`
Sets the internal status to `true`
- virtual `WaitResult Wait (int time_out=-1)`
Performs a wait operation with the object to get it.
- `bool IsValid ()`
Returns `true` if the object is valid, that is, the internal status value is `true`.
- virtual `bool Dispose ()`
Release the resources associated to the IPC object and sets the internal status to `false`.
- virtual `~IPCObject ()`
The destructor calls the method `Dispose`.

Private Attributes

- `bool valid`
Internal status of the object.

6.37.1 Detailed Description

Class base of all the IPC classes that has the basic operations (`Init`, `Wait` and `Dispose`) to be overloaded.

It has also an internal boolean value to set the object status.

For the IPC objects the Windows IPC philosophy has been adopted because of its simplicity and flexibility. Under this philosophy, the main operation that can be performed to an IPC object is `Wait`, to wait for getting the control of the object. Depending on the type of the IPC object (mutex, event, etc.), the meaning of "getting" the control of the object can be different.

6.37.2 Member Typedef Documentation

6.37.2.1 typedef `SHARED_PTR<IPCObject> ipc::IPCObject::Ptr`

Pointer to an IPC object.

6.37.3 Constructor & Destructor Documentation

6.37.3.1 `ipc::IPCObject::IPCObject () [inline]`

Initializes the internal status to `false`.

6.37.3.2 `virtual ipc::IPCObject::~~IPCObject () [inline],[virtual]`

The destructor calls the method `Dispose`.

Here is the call graph for this function:

6.37.4 Member Function Documentation

6.37.4.1 `virtual bool ipc::IPCObject::Dispose () [inline], [virtual]`

Release the resources associated to the IPC object and sets the internal status to `false`.

Returns

`true` if successful. If it is not overloaded, it always returns `true`.

Reimplemented in [ipc::Event](#), [ipc::Mutex](#), and [ipc::RdWrLock](#).

Here is the caller graph for this function:

6.37.4.2 `virtual bool ipc::IPCObject::Init () [inline], [virtual]`

Sets the internal status to `true`

Returns

`true` if successful. If it is not overloaded, it always returns `true` .

Reimplemented in [ipc::Event](#), [ipc::Mutex](#), and [ipc::RdWrLock](#).

Here is the caller graph for this function:

6.37.4.3 `bool ipc::IPCObject::IsValid () [inline]`

Returns `true` if the object is valid, that is, the internal status value is `true`.

Here is the caller graph for this function:

6.37.4.4 `virtual WaitResult ipc::IPCObject::Wait (int time_out = -1) [inline], [virtual]`

Performs a wait operation with the object to get it.

Parameters

<i>time_out</i>	Time out (infinite by default).
-----------------	---------------------------------

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error. If it is not overloaded, it always returns `WAIT_ERROR`.

Reimplemented in [ipc::Event](#), [ipc::Mutex](#), and [ipc::RdWrLock](#).

6.37.5 Member Data Documentation

6.37.5.1 `bool ipc::IPCObject::valid [private]`

Internal status of the object.

As it is a private member, the derived classes must use the methods `Init` and `Dispose` to set the value of this status.

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

- [ipc/ipc_object.h](#)

6.38 data::LockedAccess Struct Reference

Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.

```
#include <file.h>
```

Collaboration diagram for data::LockedAccess:

Static Public Member Functions

- static void [configure](#) (`FILE *file_ptr`)
- static `size_t` [fwrite](#) (`const void *ptr`, `size_t size`, `size_t count`, `FILE *file_ptr`)
- static `size_t` [fread](#) (`void *ptr`, `size_t size`, `size_t count`, `FILE *file_ptr`)
- static `int` [fgetc](#) (`FILE *file_ptr`)
- static `int` [fputc](#) (`int c`, `FILE *file_ptr`)

6.38.1 Detailed Description

Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.

See also

[File](#)

6.38.2 Member Function Documentation

6.38.2.1 static void data::LockedAccess::configure (`FILE * file_ptr`) `[inline]`, `[static]`

6.38.2.2 static int data::LockedAccess::fgetc (`FILE * file_ptr`) `[inline]`, `[static]`

6.38.2.3 static int data::LockedAccess::fputc (`int c`, `FILE * file_ptr`) `[inline]`, `[static]`

6.38.2.4 static `size_t` data::LockedAccess::fread (`void * ptr`, `size_t size`, `size_t count`, `FILE * file_ptr`) `[inline]`, `[static]`

6.38.2.5 static `size_t` data::LockedAccess::fwrite (`const void * ptr`, `size_t size`, `size_t count`, `FILE * file_ptr`) `[inline]`, `[static]`

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

- data/[file.h](#)

6.39 jpeg2000::Metadata Class Reference

Contains the indexing information associated to the meta-data of a JPEG2000 image file.

```
#include <meta_data.h>
```

Collaboration diagram for jpeg2000::Metadata:

Public Member Functions

- [Metadata](#) ()
Empty constructor.

- [Metadata](#) (const [Metadata](#) &info)
Copy constructor.
- template<typename T >
T & [SerializeWith](#) (T &stream)
- [Metadata](#) & [operator=](#) (const [Metadata](#) &info)
Copy assignment.
- virtual [~Metadata](#) ()

Public Attributes

- vector< [FileSegment](#) > [meta_data](#)
File segments of all the meta-data blocks.
- vector< [Placeholder](#) > [place_holders](#)
Associated place-holders.

Friends

- ostream & [operator<<](#) (ostream &out, const [Metadata](#) &info)

6.39.1 Detailed Description

Contains the indexing information associated to the meta-data of a JPEG2000 image file.

This class can be printed and serialized.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 `jpeg2000::Metadata::Metadata ()` `[inline]`

Empty constructor.

6.39.2.2 `jpeg2000::Metadata::Metadata (const Metadata & info)` `[inline]`

Copy constructor.

6.39.2.3 `virtual jpeg2000::Metadata::~~Metadata ()` `[inline]`, `[virtual]`

6.39.3 Member Function Documentation

6.39.3.1 `Metadata& jpeg2000::Metadata::operator= (const Metadata & info)` `[inline]`

Copy assignment.

Here is the call graph for this function:

6.39.3.2 `template<typename T > T& jpeg2000::Metadata::SerializeWith (T & stream)` `[inline]`

6.39.4 Friends And Related Function Documentation

6.39.4.1 `ostream& operator<< (ostream & out, const Metadata & info)` `[friend]`

6.39.5 Member Data Documentation

6.39.5.1 vector<FileSegment> jpeg2000::Metadata::meta_data

File segments of all the meta-data blocks.

6.39.5.2 vector<Placeholder> jpeg2000::Metadata::place_holders

Associated place-holders.

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

- [jpeg2000/meta_data.h](#)

6.40 ipc::Mutex Class Reference

IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API.

```
#include <mutex.h>
```

Inheritance diagram for ipc::Mutex:

Collaboration diagram for ipc::Mutex:

Public Types

- typedef [SHARED_PTR< Mutex > Ptr](#)
Pointer to a [Mutex](#) object.

Public Member Functions

- virtual bool [Init](#) ()
Initializes the object without locking the mutex.
- bool [Init](#) (bool initial_owner)
Initializes the object.
- virtual [WaitResult Wait](#) (int time_out=-1)
Performs a wait operation with the object to get it.
- virtual bool [Dispose](#) ()
*Release the resources associated to the IPC object and sets the internal status to *false*.*
- bool [Release](#) ()
Releases/unlocks the mutex.

Private Attributes

- pthread_t [locker](#)
Id. of the thread that locks the mutex.
- pthread_mutex_t [mutex](#)
[Mutex](#) information.

6.40.1 Detailed Description

IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API.

See also

[IPCObject](#)

6.40.2 Member Typedef Documentation

6.40.2.1 typedef SHARED_PTR<Mutex> ipc::Mutex::Ptr

Pointer to a [Mutex](#) object.

6.40.3 Member Function Documentation

6.40.3.1 bool ipc::Mutex::Dispose () [virtual]

Release the resources associated to the IPC object and sets the internal status to `false`.

Returns

`true` if successful.

Reimplemented from [ipc::IPCObject](#).

Here is the call graph for this function:

6.40.3.2 virtual bool ipc::Mutex::Init () [inline],[virtual]

Initializes the object without locking the mutex.

Returns

`true` if successful.

Reimplemented from [ipc::IPCObject](#).

Here is the caller graph for this function:

6.40.3.3 bool ipc::Mutex::Init (bool *initial_owner*)

Initializes the object.

Parameters

<i>initial_owner</i>	If <code>true</code> the mutex is locked.
----------------------	---

Returns

`true` if successful.

Here is the call graph for this function:

6.40.3.4 bool ipc::Mutex::Release ()

Releases/unlocks the mutex.

Returns

true if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.40.3.5 WaitResult ipc::Mutex::Wait (int *time_out* = -1) [virtual]

Performs a wait operation with the object to get it.

Parameters

<i>time_out</i>	Time out (infinite by default).
-----------------	---------------------------------

Returns

WAIT_OBJECT if successful, WAIT_TIMEOUT if time out or WAIT_ERROR is error.

Reimplemented from [ipc::IPCObject](#).

Here is the call graph for this function:

Here is the caller graph for this function:

6.40.4 Member Data Documentation

6.40.4.1 pthread_t ipc::Mutex::locker [private]

Id. of the thread that locks the mutex.

6.40.4.2 pthread_mutex_t ipc::Mutex::mutex [private]

[Mutex](#) information.

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

- [ipc/mutex.h](#)
- [ipc/mutex.cc](#)

6.41 data::OutputOperator Struct Reference

This struct identifies a basic output operator to be applied to a `File` object.

```
#include <serialize.h>
```

Collaboration diagram for data::OutputOperator:

Static Public Member Functions

- static const char * [FileAccess](#) ()
Returns the required file access for this operator.
- static bool [SerializeBytes](#) ([File](#) &file, void *ptr, int num_bytes)
Performs an output (write) serialization of bytes for a file.

6.41.1 Detailed Description

This struct identifies a basic output operator to be applied to a `File` object.

See also

[BaseStream](#)
[File](#)

6.41.2 Member Function Documentation

6.41.2.1 `static const char* data::OutputOperator::FileAccess () [inline],[static]`

Returns the required file access for this operator.

6.41.2.2 `static bool data::OutputOperator::SerializeBytes (File & file, void * ptr, int num_bytes) [inline],[static]`

Performs an output (write) serialization of bytes for a file.

Parameters

<i>file</i>	File to use for the operation.
<i>ptr</i>	Pointer to the buffer where to read the bytes.
<i>num_bytes</i>	Number of bytes to write to the file.

Returns

`true` if successful.

Here is the call graph for this function:

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

- [data/serialize.h](#)

6.42 data::OutputStream Class Reference

Specialization of the [BaseStream](#) for output serializations.

```
#include <serialize.h>
```

Inheritance diagram for `data::OutputStream`:

Collaboration diagram for `data::OutputStream`:

Public Member Functions

- `template<typename T>`
[OutputStream](#) & [Serialize](#) (T &var)

Additional Inherited Members

6.42.1 Detailed Description

Specialization of the [BaseStream](#) for output serializations.

See also

[BaseStream](#)

6.42.2 Member Function Documentation

6.42.2.1 `template<typename T> OutputStream& data::OutputStream::Serialize (T & var) [inline]`

Here is the call graph for this function:

Here is the caller graph for this function:

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

- data/[serialize.h](#)

6.43 jpeg2000::Packet Class Reference

Contains the information of a packet.

```
#include <packet.h>
```

Collaboration diagram for jpeg2000::Packet:

Public Member Functions

- [Packet](#) ()
Initializes the object to zero.
- [Packet](#) (int [layer](#), int [resolution](#), int [component](#), [Point](#) [precinct_xy](#))
Initializes the object.
- [Packet](#) (const [Packet](#) &packet)
Copy constructor.
- const [Packet](#) & [operator=](#) (const [Packet](#) &packet)
Copy assignment.
- virtual [~Packet](#) ()

Public Attributes

- int [layer](#)
Quality layer.
- int [component](#)
Component number.
- int [resolution](#)
Resolution level.
- [Point](#) [precinct_xy](#)
Precinct coordinate.

Friends

- ostream & [operator<<](#) (ostream &out, const [Packet](#) &packet)

6.43.1 Detailed Description

Contains the information of a packet.

This class can be printed.

6.43.2 Constructor & Destructor Documentation

6.43.2.1 `jpeg2000::Packet::Packet ()` [inline]

Initializes the object to zero.

6.43.2.2 `jpeg2000::Packet::Packet (int layer, int resolution, int component, Point precinct_xy)` [inline]

Initializes the object.

6.43.2.3 `jpeg2000::Packet::Packet (const Packet & packet)` [inline]

Copy constructor.

6.43.2.4 `virtual jpeg2000::Packet::~~Packet ()` [inline],[virtual]

6.43.3 Member Function Documentation

6.43.3.1 `const Packet& jpeg2000::Packet::operator= (const Packet & packet)` [inline]

Copy assignment.

6.43.4 Friends And Related Function Documentation

6.43.4.1 `ostream& operator<< (ostream & out, const Packet & packet)` [friend]

6.43.5 Member Data Documentation

6.43.5.1 `int jpeg2000::Packet::component`

Component number.

6.43.5.2 `int jpeg2000::Packet::layer`

Quality layer.

6.43.5.3 `Point jpeg2000::Packet::precinct_xy`

Precinct coordinate.

6.43.5.4 `int jpeg2000::Packet::resolution`

Resolution level.

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

- [jpeg2000/packet.h](#)

6.44 jpeg2000::PacketIndex Class Reference

Class used for indexing the packets of a codestream image.

```
#include <packet_index.h>
```

Collaboration diagram for jpeg2000::PacketIndex:

Public Types

- enum { [MINIMUM_OFFSET](#) = 64 }

Public Member Functions

- [PacketIndex](#) ()
Empty constructor.
- [PacketIndex](#) (uint64_t max_offset)
Initializes the object.
- [PacketIndex](#) (const [PacketIndex](#) &index)
Copy constructor.
- const [PacketIndex](#) & [operator=](#) (const [PacketIndex](#) &index)
Copy assignment.
- [PacketIndex](#) & [Add](#) (const [FileSegment](#) &segment)
Adds a new packet segment to the index.
- int [Size](#) () const
Returns the number of elements of the vector.
- void [Clear](#) ()
Clears the content.
- [FileSegment](#) [operator\[\]](#) (int i) const
Operator used for accessing the items.
- virtual [~PacketIndex](#) ()

Private Attributes

- [vint_vector](#) offsets
Vector of packet offsets.
- vector< [FileSegment](#) > aux
Vector of file segments to handle the different sets of packets that are not contiguous.

6.44.1 Detailed Description

Class used for indexing the packets of a codestream image.

The class `vint_vector` is used internally to store the offsets of the packets with the minimum required bytes.

See also

[data::vint_vector](#)

6.44.2 Member Enumeration Documentation

6.44.2.1 anonymous enum

Enumerator

MINIMUM_OFFSET All the offsets must be greater than this value.

6.44.3 Constructor & Destructor Documentation

6.44.3.1 jpeg2000::PacketIndex::PacketIndex () [inline]

Empty constructor.

6.44.3.2 jpeg2000::PacketIndex::PacketIndex (uint64_t *max_offset*) [inline]

Initializes the object.

Parameters

<i>max_offset</i>	Maximum value for an offset.
-------------------	------------------------------

Here is the call graph for this function:

6.44.3.3 jpeg2000::PacketIndex::PacketIndex (const PacketIndex & *index*) [inline]

Copy constructor.

6.44.3.4 virtual jpeg2000::PacketIndex::~~PacketIndex () [inline],[virtual]

6.44.4 Member Function Documentation

6.44.4.1 PacketIndex& jpeg2000::PacketIndex::Add (const FileSegment & *segment*) [inline]

Adds a new packet segment to the index.

Parameters

<i>segment</i>	Fiel segment associated to the packet.
----------------	--

Returns

The object itself.

Here is the call graph for this function:

6.44.4.2 void jpeg2000::PacketIndex::Clear () [inline]

Clears the content.

Here is the call graph for this function:

6.44.4.3 const PacketIndex& jpeg2000::PacketIndex::operator= (const PacketIndex & *index*) [inline]

Copy assignment.

6.44.4.4 FileSegment jpeg2000::PacketIndex::operator[] (int i) const [inline]

Operator used for accessing the items.

Parameters

<i>i</i>	Item index.
----------	-------------

Returns

File segment of the packet.

6.44.4.5 `int jpeg2000::PacketIndex::Size () const` `[inline]`

Returns the number of elements of the vector.

Here is the call graph for this function:

6.44.5 Member Data Documentation

6.44.5.1 `vector<FileSegment> jpeg2000::PacketIndex::aux` `[private]`

Vector of file segments to handle the different sets of packets that are not contiguous.

6.44.5.2 `vint_vector jpeg2000::PacketIndex::offsets` `[private]`

Vector of packet offsets.

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

- [jpeg2000/packet_index.h](#)

6.45 jpip::Request::ParametersMask Union Reference

Union used to control the presence of the different JPIP parameters in a request.

```
#include <request.h>
```

Collaboration diagram for `jpip::Request::ParametersMask`:

Public Member Functions

- [ParametersMask](#) ()
Initializes the mask to zero.
- `bool HasWOI () const`
Returns `true` if the mask contains the parameters associated to the [WOI](#) (`fsiz`, `roff` and `rsiz`).
- `void Clear ()`
Sets the mask to zero.

Public Attributes

- `struct {`
 - `int fsiz: 1`
 - `int roff: 1`
 - `int rsiz: 1`
 - `int metareq: 1`
 - `int len: 1`

```

int target: 1
int cid: 1
int cnew: 1
int cclose: 1
    int model: 1
    int stream: 1
    int context: 1
} items

```

Parameters mask.

- int [value](#)

Parameters mask as integer.

6.45.1 Detailed Description

Union used to control the presence of the different JPIP parameters in a request.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 `jipip::Request::ParametersMask::ParametersMask ()` `[inline]`

Initializes the mask to zero.

6.45.3 Member Function Documentation

6.45.3.1 `void jipip::Request::ParametersMask::Clear ()` `[inline]`

Sets the mask to zero.

Here is the caller graph for this function:

6.45.3.2 `bool jipip::Request::ParametersMask::HasWOI () const` `[inline]`

Returns `true` if the mask contains the parameters associated to the [WOI](#) (fsiz, roff and rsiz).

Here is the caller graph for this function:

6.45.4 Member Data Documentation

6.45.4.1 `int jipip::Request::ParametersMask::cclose`

6.45.4.2 `int jipip::Request::ParametersMask::cid`

6.45.4.3 `int jipip::Request::ParametersMask::cnew`

6.45.4.4 `int jipip::Request::ParametersMask::context`

6.45.4.5 `int jipip::Request::ParametersMask::fsiz`

6.45.4.6 `struct { ... } jipip::Request::ParametersMask::items`

Parameters mask.

- 6.45.4.7 `int jpip::Request::ParametersMask::len`
- 6.45.4.8 `int jpip::Request::ParametersMask::metareq`
- 6.45.4.9 `int jpip::Request::ParametersMask::model`
- 6.45.4.10 `int jpip::Request::ParametersMask::roff`
- 6.45.4.11 `int jpip::Request::ParametersMask::rsiz`
- 6.45.4.12 `int jpip::Request::ParametersMask::stream`
- 6.45.4.13 `int jpip::Request::ParametersMask::target`
- 6.45.4.14 `int jpip::Request::ParametersMask::value`

Parameters mask as integer.

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

- [jpip/request.h](#)

6.46 jpeg2000::Placeholder Class Reference

Contains the information of a place-holder.

```
#include <place_holder.h>
```

Collaboration diagram for jpeg2000::Placeholder:

Public Member Functions

- [Placeholder](#) ()
Initializes the object.
- [Placeholder](#) (int `id`, bool `is_jp2c`, const [FileSegment](#) &`header`, uint64_t `data_length`)
Initializes the object.
- [Placeholder](#) (const [Placeholder](#) &`place_holder`)
Copy constructor.
- `template<typename T >`
 T & [SerializeWith](#) (T &`stream`)
- [Placeholder](#) & `operator=` (const [Placeholder](#) &`place_holder`)
Copy assignment.
- int [length](#) () const
Returns the length of the place-holder.
- virtual `~Placeholder` ()

Public Attributes

- int `id`
Place-holder identifier.
- bool `is_jp2c`
true if refers to a codestream.
- [FileSegment](#) `header`

File segment associated to the box header.

- uint64_t [data_length](#)

Length of the place-holder data.

Friends

- ostream & [operator<<](#) (ostream &out, const [Placeholder](#) &place_holder)

6.46.1 Detailed Description

Contains the information of a place-holder.

This class can be printed and serialized.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 jpeg2000::Placeholder::Placeholder () `[inline]`

Initializes the object.

6.46.2.2 jpeg2000::Placeholder::Placeholder (int *id*, bool *is_jp2c*, const FileSegment & *header*, uint64_t *data_length*) `[inline]`

Initializes the object.

Parameters

<i>id</i>	Place-holder identifier.
<i>is_jp2c</i>	Indicates if is a codestream place-holder.
<i>header</i>	File segment of the associated header.
<i>data_length</i>	Length of the place-holder data.

6.46.2.3 jpeg2000::Placeholder::Placeholder (const Placeholder & *place_holder*) `[inline]`

Copy constructor.

6.46.2.4 virtual jpeg2000::Placeholder::~~Placeholder () `[inline], [virtual]`

6.46.3 Member Function Documentation

6.46.3.1 int jpeg2000::Placeholder::length () const `[inline]`

Returns the length of the place-holder.

Here is the caller graph for this function:

6.46.3.2 Placeholder& jpeg2000::Placeholder::operator= (const Placeholder & *place_holder*) `[inline]`

Copy assignment.

6.46.3.3 `template<typename T> T& jpeg2000::Placeholder::SerializeWith (T & stream)` `[inline]`

6.46.4 Friends And Related Function Documentation

6.46.4.1 `ostream& operator<< (ostream & out, const Placeholder & place_holder)` `[friend]`

6.46.5 Member Data Documentation

6.46.5.1 `uint64_t jpeg2000::Placeholder::data_length`

Length of the place-holder data.

6.46.5.2 `FileSegment jpeg2000::Placeholder::header`

File segment associated to the box header.

6.46.5.3 `int jpeg2000::Placeholder::id`

Place-holder identifier.

6.46.5.4 `bool jpeg2000::Placeholder::is_jp2c`

`true` if refers to a codestream.

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

- [jpeg2000/place_holder.h](#)

6.47 jpeg2000::Point Class Reference

Represents a couple of integer values that can be used to identify a coordinate as well as a size.

```
#include <point.h>
```

Collaboration diagram for jpeg2000::Point:

Public Member Functions

- [Point](#) ()
Initializes the object.
- [Point](#) (int *x*, int *y*)
Initializes the object.
- [Point](#) (const [Point](#) &*p*)
Copy constructor.
- [Point](#) & [operator=](#) (const [Point](#) &*p*)
Copy assignment.
- [Point](#) & [operator++](#) ()
Increments by one the two values.
- [Point](#) & [operator--](#) ()
Decrements by one the two values.
- [Point](#) & [operator+=](#) (int *val*)
Increments the two values.

- [Point](#) & [operator--](#) (int val)
Decrements the two values.
- [Point](#) & [operator*=](#) (int val)
Multiplies the two values by one value.
- [Point](#) & [operator/=](#) (int val)
Divides the two values by one value.
- template<typename T >
T & [SerializeWith](#) (T &stream)
- virtual [~Point](#) ()

Public Attributes

- int [x](#)
Value X.
- int [y](#)
Value Y.

Friends

- [Point](#) [operator+](#) (const [Point](#) &a, int value)
Returns the sum of a point with an integer value.
- [Point](#) [operator-](#) (const [Point](#) &a, int value)
Returns the subtraction of a point with an integer value.
- [Point](#) [operator*](#) (const [Point](#) &a, int value)
Returns the multiplication of a point with an integer value.
- [Point](#) [operator/](#) (const [Point](#) &a, int value)
Returns the division of a point with an integer value.
- [Point](#) [operator+](#) (const [Point](#) &a, const [Point](#) &b)
Returns the sum of two points.
- [Point](#) [operator-](#) (const [Point](#) &a, const [Point](#) &b)
Returns the subtraction of two points.
- [Point](#) [operator*](#) (const [Point](#) &a, const [Point](#) &b)
Returns the multiplication of two points.
- [Point](#) [operator/](#) (const [Point](#) &a, const [Point](#) &b)
Returns the division of two points.
- bool [operator==](#) (const [Point](#) &a, const [Point](#) &b)
Returns `true` if the two points are equal.
- bool [operator!=](#) (const [Point](#) &a, const [Point](#) &b)
Returns `true` if the two points are not equal.
- ostream & [operator<<](#) (ostream &out, const [Point](#) &point)

6.47.1 Detailed Description

Represents a couple of integer values that can be used to identify a coordinate as well as a size.

This class can be printed and serialized.

6.47.2 Constructor & Destructor Documentation

6.47.2.1 jpeg2000::Point::Point () `[inline]`

Initializes the object.

6.47.2.2 `jpeg2000::Point::Point (int x, int y) [inline]`

Initializes the object.

Parameters

<i>x</i>	Value X.
<i>y</i>	Value Y.

6.47.2.3 jpeg2000::Point::Point (const Point & *p*) [inline]

Copy constructor.

6.47.2.4 virtual jpeg2000::Point::~~Point () [inline],[virtual]**6.47.3 Member Function Documentation****6.47.3.1** Point& jpeg2000::Point::operator*=(int *val*) [inline]

Multiplies the two values by one value.

Parameters

<i>val</i>	Value to multiply.
------------	--------------------

Returns

The object itself.

6.47.3.2 Point& jpeg2000::Point::operator++ () [inline]

Increments by one the two values.

Returns

The object itself.

6.47.3.3 Point& jpeg2000::Point::operator+=(int *val*) [inline]

Increments the two values.

Parameters

<i>val</i>	Value to increment.
------------	---------------------

Returns

The object itself.

6.47.3.4 Point& jpeg2000::Point::operator-- () [inline]

Decrements by one the two values.

Returns

The object itself.

6.47.3.5 Point& jpeg2000::Point::operator-=(int *val*) [inline]

Decrements the two values.

Parameters

<i>val</i>	Value to decrement.
------------	---------------------

Returns

The object itself.

6.47.3.6 **Point& jpeg2000::Point::operator/=(int *val*)** `[inline]`

Divides the two values by one value.

Parameters

<i>val</i>	Value to divide.
------------	------------------

Returns

The object itself.

6.47.3.7 **Point& jpeg2000::Point::operator=(const **Point** & *p*)** `[inline]`

Copy assignment.

6.47.3.8 **template<typename T> T& jpeg2000::Point::SerializeWith (T & *stream*)** `[inline]`

6.47.4 Friends And Related Function Documentation

6.47.4.1 **bool operator!=(const **Point** & *a*, const **Point** & *b*)** `[friend]`

Returns `true` if the two points are not equal.

6.47.4.2 **Point operator*(const **Point** & *a*, int *value*)** `[friend]`

Returns the multiplication of a point with an integer value.

The value is multiplied to the two values of the point.

6.47.4.3 **Point operator*(const **Point** & *a*, const **Point** & *b*)** `[friend]`

Returns the multiplication of two points.

The operation is applied each value of each point.

6.47.4.4 **Point operator+ (const **Point** & *a*, int *value*)** `[friend]`

Returns the sum of a point with an integer value.

The value is added to the two values of the point.

6.47.4.5 **Point operator+ (const **Point** & *a*, const **Point** & *b*)** `[friend]`

Returns the sum of two points.

The operation is applied each value of each point.

6.47.4.6 Point operator- (const Point & a, int value) [friend]

Returns the subtraction of a point with an integer value.

The value is subtracted from the two values of the point.

6.47.4.7 Point operator- (const Point & a, const Point & b) [friend]

Returns the subtraction of two points.

The operation is applied each value of each point.

6.47.4.8 Point operator/ (const Point & a, int value) [friend]

Returns the division of a point with an integer value.

The value is divided to the two values of the point.

6.47.4.9 Point operator/ (const Point & a, const Point & b) [friend]

Returns the division of two points.

The operation is applied each value of each point.

6.47.4.10 ostream& operator<< (ostream & out, const Point & point) [friend]**6.47.4.11 bool operator== (const Point & a, const Point & b) [friend]**

Returns `true` if the two points are equal.

6.47.5 Member Data Documentation**6.47.5.1 int jpeg2000::Point::x**

Value X.

6.47.5.2 int jpeg2000::Point::y

Value Y.

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

- [jpeg2000/point.h](#)

6.48 net::PollFD Struct Reference

Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.

```
#include <poll_table.h>
```

Inheritance diagram for `net::PollFD`:

Collaboration diagram for `net::PollFD`:

Public Member Functions

- [PollFD](#) (int vfd, int mask)
Initializes the structure.
- bool [operator==](#) (int n)
Returns `true` if the file descriptor is the same as the given value.

6.48.1 Detailed Description

Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.

See also

[PollTable](#)

6.48.2 Constructor & Destructor Documentation

6.48.2.1 `net::PollFD::PollFD (int vfd, int mask) [inline]`

Initializes the structure.

Parameters

<i>vfd</i>	File descriptor.
<i>mask</i>	Poll mask.

6.48.3 Member Function Documentation

6.48.3.1 `bool net::PollFD::operator== (int n) [inline]`

Returns `true` if the file descriptor is the same as the given value.

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

- [net/poll_table.h](#)

6.49 net::PollTable Class Reference

This class allows to perform polls easily over a vector of descriptors.

```
#include <poll_table.h>
```

Collaboration diagram for `net::PollTable`:

Public Member Functions

- [PollTable](#) ()
- void [Add](#) (int fd, int mask)
Adds a new file descriptor and mask to the vector.
- int [Poll](#) (int timeout=-1)
Performs a poll over all the descriptors using the associated masks.
- int [GetSize](#) () const
Returns the size of the internal vector.
- void [Remove](#) (int fd)

- Removes an item of the internal vector giving its file descriptor.*
- void [RemoveAt](#) (int n)
Remove an item of the internal vector giving its index position.
- [PollFD](#) & [operator\[\]](#) (int n)
Indexing operator.
- virtual [~PollTable](#) ()

Private Attributes

- vector< [PollFD](#) > [fds](#)
Vector with the file descriptors and masks for polling.

6.49.1 Detailed Description

This class allows to perform polls easily over a vector of descriptors.

It uses an internal STL vector of [PollFD](#) objects to handle dinamically the file descriptors and masks.

See also

[PollFD](#)

6.49.2 Constructor & Destructor Documentation

6.49.2.1 `net::PollTable::PollTable ()` `[inline]`

6.49.2.2 `virtual net::PollTable::~~PollTable ()` `[inline]`, `[virtual]`

6.49.3 Member Function Documentation

6.49.3.1 `void net::PollTable::Add (int fd, int mask)` `[inline]`

Adds a new file descriptor and mask to the vector.

Parameters

<i>fd</i>	File descriptor.
<i>mask</i>	Polling mask.

Here is the caller graph for this function:

6.49.3.2 `int net::PollTable::GetSize () const` `[inline]`

Returns the size of the internal vector.

Here is the caller graph for this function:

6.49.3.3 `PollFD& net::PollTable::operator[] (int n)` `[inline]`

Indexing operator.

6.49.3.4 `int net::PollTable::Poll (int timeout = -1)` `[inline]`

Peforms a poll over all the descriptors using the associated masks.

Parameters

<i>timeout</i>	Time out of the poll (infinite by default).
----------------	---

Returns

The value given by the kernel function `poll`.

Here is the caller graph for this function:

6.49.3.5 void net::PollTable::Remove (int *fd*) [inline]

Removes an item of the internal vector giving its file descriptor.

Parameters

<i>fd</i>	File descriptor to remove.
-----------	----------------------------

Here is the caller graph for this function:

6.49.3.6 void net::PollTable::RemoveAt (int *n*) [inline]

Remove an item of the internal vector giving its index position.

Parameters

<i>n</i>	Position of the item to remove.
----------	---------------------------------

Here is the caller graph for this function:

6.49.4 Member Data Documentation**6.49.4.1 vector<PollFD> net::PollTable::fds [private]**

Vector with the file descriptors and masks for polling.

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

- [net/poll_table.h](#)

6.50 http::Protocol Class Reference

Class used to identify the HTTP protocol.

```
#include <protocol.h>
```

Collaboration diagram for `http::Protocol`:

Public Member Functions

- [Protocol](#) (int [majorVersion](#)=1, int [minorVersion](#)=1)
Initialized the protocol with the given version.
- [Protocol](#) (const [Protocol](#) &protocol)
Copy constructor.
- int [majorVersion](#) () const
Returns the mayor number of the protocol version.
- int [minorVersion](#) () const
Returns the minor number of the protocol version.

Static Public Attributes

- static const char `CRLF` [] = "\r\n"

String with the characters 13 (CR) and 10 (LF), the line separator used in the HTTP protocol.

Private Attributes

- int `mayorVersion_`

Mayor protocol version.

- int `minorVersion_`

Minor protocol version.

Friends

- ostream & `operator<<` (ostream &out, const `Protocol` &protocol)
- istream & `operator>>` (istream &in, `Protocol` &protocol)

6.50.1 Detailed Description

Class used to identify the HTTP protocol.

It is possible to use this class with standard streams.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 `http::Protocol::Protocol (int mayorVersion = 1, int minorVersion = 1)` `[inline]`

Initialized the protocol with the given version.

By default the version is 1.1.

Parameters

<i>mayorVersion</i>	Mayor protocol version
<i>minorVersion</i>	Minor protocol version

6.50.2.2 `http::Protocol::Protocol (const Protocol & protocol)` `[inline]`

Copy constructor.

6.50.3 Member Function Documentation

6.50.3.1 `int http::Protocol::mayorVersion () const` `[inline]`

Returns the mayor number of the protocol version.

6.50.3.2 `int http::Protocol::minorVersion () const` `[inline]`

Returns the minor number of the protocol version.

6.50.4 Friends And Related Function Documentation

6.50.4.1 `ostream& operator<< (ostream & out, const Protocol & protocol) [friend]`

6.50.4.2 `istream& operator>> (istream & in, Protocol & protocol) [friend]`

6.50.5 Member Data Documentation

6.50.5.1 `const char http::Protocol::CRLF = "\r\n" [static]`

String with the characters 13 (CR) and 10 (LF), the line separator used in the HTTP protocol.

6.50.5.2 `int http::Protocol::majorVersion_ [private]`

Mayor protocol version.

6.50.5.3 `int http::Protocol::minorVersion_ [private]`

Minor protocol version.

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

- [http/protocol.h](#)
- [http/protocol.cc](#)

6.51 jpeg2000::Range Class Reference

Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

```
#include <range.h>
```

Collaboration diagram for jpeg2000::Range:

Public Member Functions

- [Range](#) ()
Initializes the object.
- [Range](#) (int first, int last)
Initializes the object.
- [Range](#) (const [Range](#) &range)
Copy constructor.
- [Range](#) & [operator=](#) (const [Range](#) &range)
Copy assignment.
- bool [IsValid](#) () const
Returns true if the first value is greater or equal to zero, and it is less or equal to the last value.
- int [GetItem](#) (int i) const
Returns an item of the range, starting at the first value.
- int [GetIndex](#) (int item) const
Returns the index of an item of the range.
- int [Length](#) () const
Returns the length of the range (last - first + 1).
- virtual [~Range](#) ()

Public Attributes

- int [first](#)
First value of the range.
- int [last](#)
Last value of the range.

Friends

- bool [operator==](#) (const [Range](#) &a, const [Range](#) &b)
- bool [operator!=](#) (const [Range](#) &a, const [Range](#) &b)
- ostream & [operator<<](#) (ostream &out, const [Range](#) &range)

6.51.1 Detailed Description

Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

Some basic operations are defined for easing the work with ranges.

6.51.2 Constructor & Destructor Documentation

6.51.2.1 jpeg2000::Range::Range () [\[inline\]](#)

Initializes the object.

6.51.2.2 jpeg2000::Range::Range (int *first*, int *last*) [\[inline\]](#)

Initializes the object.

Parameters

<i>first</i>	First value.
<i>last</i>	Last value.

6.51.2.3 jpeg2000::Range::Range (const [Range](#) &range) [\[inline\]](#)

Copy constructor.

6.51.2.4 virtual jpeg2000::Range::~Range () [\[inline\]](#), [\[virtual\]](#)

6.51.3 Member Function Documentation

6.51.3.1 int jpeg2000::Range::GetIndex (int *item*) const [\[inline\]](#)

Returns the index of an item of the range.

Parameters

<i>item</i>	Item of the range.
-------------	--------------------

Returns

item - first.

Here is the caller graph for this function:

6.51.3.2 `int jpeg2000::Range::GetItem (int i) const` `[inline]`

Returns an item of the range, starting at the first value.

Parameters

<i>i</i>	Item index.
----------	-------------

Returns

first + i.

Here is the caller graph for this function:

6.51.3.3 `bool jpeg2000::Range::IsValid () const` `[inline]`

Returns `true` if the first value is greater or equal to zero, and it is less or equal to the last value.

6.51.3.4 `int jpeg2000::Range::Length () const` `[inline]`

Returns the length of the range (last - first + 1).

Here is the caller graph for this function:

6.51.3.5 `Range& jpeg2000::Range::operator= (const Range & range)` `[inline]`

Copy assignment.

6.51.4 Friends And Related Function Documentation

6.51.4.1 `bool operator!= (const Range & a, const Range & b)` `[friend]`

6.51.4.2 `ostream& operator<< (ostream & out, const Range & range)` `[friend]`

6.51.4.3 `bool operator== (const Range & a, const Range & b)` `[friend]`

6.51.5 Member Data Documentation

6.51.5.1 `int jpeg2000::Range::first`

First value of the range.

6.51.5.2 `int jpeg2000::Range::last`

Last value of the range.

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

- [jpeg2000/range.h](#)

6.52 ipc::RdWrLock Class Reference

IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

```
#include <rdwr_lock.h>
```

Inheritance diagram for ipc::RdWrLock:

Collaboration diagram for ipc::RdWrLock:

Public Types

- typedef [SHARED_PTR< RdWrLock > Ptr](#)
Pointer to a [RdWrLock](#) object.

Public Member Functions

- virtual bool [Init](#) ()
Initializes the object.
- virtual [WaitResult Wait](#) (int time_out=-1)
Performs a wait operation with the object to get it for reading.
- [WaitResult WaitForWriting](#) (int time_out=-1)
Performs a wait operation with the object to get it for writing.
- virtual bool [Dispose](#) ()
*Release the resources associated to the IPC object and sets the internal status to *false*.*
- bool [Release](#) ()
Releases the lock.

Private Attributes

- pthread_rwlock_t [rwlock](#)
Read/write lock information.

6.52.1 Detailed Description

IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

See also

[IPCObject](#)

6.52.2 Member Typedef Documentation

6.52.2.1 typedef SHARED_PTR<RdWrLock> ipc::RdWrLock::Ptr

Pointer to a [RdWrLock](#) object.

6.52.3 Member Function Documentation

6.52.3.1 bool ipc::RdWrLock::Dispose () [virtual]

Release the resources associated to the IPC object and sets the internal status to *false*.

Returns

`true` if successful.

Reimplemented from [ipc::IPCObject](#).

6.52.3.2 `bool ipc::RdWrLock::Init () [virtual]`

Initializes the object.

Returns

`true` if successful.

Reimplemented from [ipc::IPCObject](#).

6.52.3.3 `bool ipc::RdWrLock::Release ()`

Releases the lock.

Returns

`true` if successful.

6.52.3.4 `WaitResult ipc::RdWrLock::Wait (int time_out = -1) [virtual]`

Performs a wait operation with the object to get it for reading.

Parameters

<i>time_out</i>	Time out (infinite by default).
-----------------	---------------------------------

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error.

Reimplemented from [ipc::IPCObject](#).

6.52.3.5 `WaitResult ipc::RdWrLock::WaitForWriting (int time_out = -1)`

Performs a wait operation with the object to get it for writing.

Parameters

<i>time_out</i>	Time out (infinite by default).
-----------------	---------------------------------

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error.

6.52.4 Member Data Documentation**6.52.4.1** `pthread_rwlock_t ipc::RdWrLock::rwlock [private]`

Read/write lock information.

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

- [ipc/rdwr_lock.h](#)
- [ipc/rdwr_lock.cc](#)

6.53 jpip::Request Class Reference

Class derived from the HTTP [Request](#) class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.

```
#include <request.h>
```

Inheritance diagram for jpip::Request:

Collaboration diagram for jpip::Request:

Classes

- union [ParametersMask](#)

Union used to control the presence of the different JPIP parameters in a request.

Public Types

- enum [RoundDirection](#) { [ROUNDUP](#), [ROUNDDOWN](#), [CLOSEST](#) }

Enumeration of the possible round directions of a [WOI](#) for specifying the resolution levels.

Public Member Functions

- [istream](#) & [ParseModel](#) ([istream](#) &stream)
Parses a cache model from an input stream.
- [istream](#) & [GetCodedChar](#) ([istream](#) &in, char &c)
Gets a coded char from an input stream.
- virtual void [ParseParameters](#) ([istream](#) &stream)
Parses the parameters of a CGI HTTP request.
- virtual void [ParseParameter](#) ([istream](#) &stream, const string ¶m, string &value)
Parses one parameter of a CGI HTTP request.
- [Request](#) ()
Empty constructor.
- void [GetResolution](#) (const [CodingParameters::Ptr](#) &coding_parameters, [WOI](#) *woi) const
Obtains the resolution level and modifies the given [WOI](#) to adjust it according to that level.
- virtual [~Request](#) ()

Public Attributes

- [Size](#) woi_size
[WOI](#) size.
- [Point](#) woi_position
[WOI](#) position.
- int [min_codestream](#)
Minimum codestream.
- int [max_codestream](#)
Maximum codestream.
- int [length_response](#)
Maximum response length.
- [ParametersMask](#) mask
Parameters mask.
- [Size](#) resolution_size

Size of the resolution level.

- [CacheModel](#) `cache_model`

Cache model.

- [RoundDirection](#) `round_direction`

Round direction.

6.53.1 Detailed Description

Class derived from the HTTP [Request](#) class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.

See also

[http::Request](#)
[CacheModel](#)

6.53.2 Member Enumeration Documentation

6.53.2.1 enum `jpip::Request::RoundDirection`

Enumeration of the possible round directions of a [WOI](#) for specifying the resolution levels.

Enumerator

ROUNDUP Round-up.

ROUNDDOWN Round-down.

CLOSEST Closest.

6.53.3 Constructor & Destructor Documentation

6.53.3.1 `jpip::Request::Request ()` `[inline]`

Empty constructor.

6.53.3.2 `virtual jpip::Request::~~Request ()` `[inline]`, `[virtual]`

6.53.4 Member Function Documentation

6.53.4.1 `istream & jpip::Request::GetCodedChar (istream & in, char & c)`

Gets a coded char from an input stream.

Parameters

<i>in</i>	Input stream.
<i>c</i>	Reference to store the char.

Returns

The same input stream.

Here is the caller graph for this function:

6.53.4.2 void jpip::Request::GetResolution (const CodingParameters::Ptr & *coding_parameters*, WOI * *woi*) const
[inline]

Obtains the resolution level and modifies the given [WOI](#) to adjust it according to that level.

Parameters

<i>coding_↔ parameters</i>	Associated coding parameters.
<i>woi</i>	WOI to modify.

Here is the caller graph for this function:

6.53.4.3 `istream & jpip::Request::ParseModel (istream & stream)`

Parses a cache model from an input stream.

Parameters

<i>stream</i>	Input stream.
---------------	---------------

Returns

The same input stream after the parsing.

Here is the call graph for this function:

Here is the caller graph for this function:

6.53.4.4 `void jpip::Request::ParseParameter (istream & stream, const string & param, string & value)` `[virtual]`

Parses one parameter of a CGI HTTP request.

Parameters

<i>stream</i>	Input stream.
<i>param</i>	String to store the parameter name.
<i>value</i>	String to store the parameter value.

Reimplemented from [http::Request](#).

Here is the call graph for this function:

6.53.4.5 `void jpip::Request::ParseParameters (istream & stream)` `[virtual]`

Parses the parameters of a CGI HTTP request.

Parameters

<i>stream</i>	Input stream.
---------------	---------------

Reimplemented from [http::Request](#).

Here is the call graph for this function:

Here is the caller graph for this function:

6.53.5 Member Data Documentation

6.53.5.1 `CacheModel jpip::Request::cache_model`

Cache model.

6.53.5.2 `int jpip::Request::length_response`

Maximum response length.

6.53.5.3 ParametersMask `jpeg::Request::mask`

Parameters mask.

6.53.5.4 `int jpeg::Request::max_codestream`

Maximum codestream.

6.53.5.5 `int jpeg::Request::min_codestream`

Minimum codestream.

6.53.5.6 `Size jpeg::Request::resolution_size`

Size of the resolution level.

6.53.5.7 RoundDirection `jpeg::Request::round_direction`

Round direction.

6.53.5.8 `Point jpeg::Request::woi_position`

[WOI](#) position.

6.53.5.9 `Size jpeg::Request::woi_size`

[WOI](#) size.

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

- [jpeg/request.h](#)
- [jpeg/request.cc](#)

6.54 http::Request Class Reference

Class used to identify a HTTP request (GET or POST).

```
#include <request.h>
```

Inheritance diagram for `http::Request`:

Collaboration diagram for `http::Request`:

Public Types

- enum [Type](#) { [GET](#), [POST](#), [UNKNOWN](#) }
- [Request](#) type enumeration.

Public Member Functions

- [Request](#) ([Type](#) type=[Request::GET](#), const string &uri="", const [Protocol](#) &protocol=[Protocol](#)(1, 1))
Initializes the request.
- bool [Parse](#) (const string &line)
Parses a request from a string.
- void [ParseURI](#) (const string &uri)
Parses a URI from a string.
- virtual void [ParseParameters](#) (istream &stream)
Parses the parameters from an input stream.
- virtual void [ParseParameter](#) (istream &stream, const string ¶m, string &value)
Parses one parameter from an input stream.

Public Attributes

- [Type](#) type
Request type (GET or POST)
- string [object](#)
Object associated to the request.
- [Protocol](#) protocol
Protocol version used.
- map< string, string > [parameters](#)
Map with all the parameters when using the CGI form.

Friends

- istream & [operator>>](#) (istream &in, [Request](#) &request)
- ostream & [operator<<](#) (ostream &out, const [Request](#) &request)

6.54.1 Detailed Description

Class used to identify a HTTP request (GET or POST).

It is possible to use this class with standard streams.

See also

[Response](#)

6.54.2 Member Enumeration Documentation

6.54.2.1 enum http::Request::Type

[Request](#) type enumeration.

Enumerator

GET

POST

UNKNOWN

6.54.3 Constructor & Destructor Documentation

6.54.3.1 `http::Request::Request (Type type = Request::GET, const string & uri = " / ", const Protocol & protocol = Protocol(1, 1)) [inline]`

Initializes the request.

Parameters

<i>type</i>	Request type (GET by default).
<i>uri</i>	Request URI ("/" by default).
<i>protocol</i>	Protocol version (1.1 by default).

6.54.4 Member Function Documentation

6.54.4.1 `bool http::Request::Parse (const string & line)`

Parses a request from a string.

Parameters

<i>line</i>	String that contains the request to parse.
-------------	--

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.54.4.2 `void http::Request::ParseParameter (istream & stream, const string & param, string & value) [virtual]`

Parses one parameter from an input stream.

Parameters

<i>stream</i>	Input stream.
<i>param</i>	Parameter name.
<i>value</i>	Parameter value.

Reimplemented in [jpip::Request](#).

Here is the caller graph for this function:

6.54.4.3 `void http::Request::ParseParameters (istream & stream) [virtual]`

Parses the parameters from an input stream.

Parameters

<i>stream</i>	Input stream.
---------------	---------------

Reimplemented in [jpip::Request](#).

Here is the call graph for this function:

Here is the caller graph for this function:

6.54.4.4 `void http::Request::ParseURI (const string & uri) [inline]`

Parses a URI from a string.

Parameters

<i>uri</i>	String that contains the URI to parse.
------------	--

Here is the caller graph for this function:

6.54.5 Friends And Related Function Documentation

6.54.5.1 `ostream& operator<< (ostream & out, const Request & request)` [[friend](#)]

6.54.5.2 `istream& operator>> (istream & in, Request & request)` [[friend](#)]

6.54.6 Member Data Documentation

6.54.6.1 `string http::Request::object`

Object associated to the request.

6.54.6.2 `map<string, string> http::Request::parameters`

Map with all the parameters when using the CGI form.

6.54.6.3 **Protocol** `http::Request::protocol`

[Protocol](#) version used.

6.54.6.4 **Type** `http::Request::type`

[Request](#) type (GET or POST)

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

- [http/request.h](#)
- [http/request.cc](#)

6.55 http::Response Class Reference

Class used to identify a HTTP response.

```
#include <response.h>
```

Collaboration diagram for `http::Response`:

Classes

- class [StatusCodesInitializer](#)
Class used for the initializer.

Public Member Functions

- [Response](#) (int `code`=200, const [Protocol](#) &`protocol`=[Protocol](#)(1, 1))
Initializes the response.

Public Attributes

- int [code](#)
Status code.
- [Protocol](#) [protocol](#)
Protocol version.

Static Public Attributes

- static map< int, string > [StatusCodes](#)
Map with the strings associated to the most commonly used status codes.

Static Private Attributes

- static [StatusCodesInitializer](#) [statusCodesInitializer](#)
The initializer of the `StatusCodes` member.

Friends

- ostream & [operator<<](#) (ostream &out, const [Response](#) &response)
- istream & [operator>>](#) (istream &in, [Response](#) &response)

6.55.1 Detailed Description

Class used to identify a HTTP response.

It is possible to use this class with standard streams.

See also

[Request](#)

6.55.2 Constructor & Destructor Documentation

6.55.2.1 `http::Response::Response (int code = 200, const Protocol & protocol = Protocol (1, 1))` `[inline]`

Initializes the response.

Parameters

<i>code</i>	Status code (200 by default).
<i>protocol</i>	Protocol version (1.1 by default).

6.55.3 Friends And Related Function Documentation

6.55.3.1 `ostream& operator<< (ostream & out, const Response & response)` `[friend]`

6.55.3.2 `istream& operator>> (istream & in, Response & response)` `[friend]`

6.55.4 Member Data Documentation

6.55.4.1 `int http::Response::code`

Status code.

6.55.4.2 Protocol `http::Response::protocol`

[Protocol](#) version.

6.55.4.3 `map< int, string > http::Response::StatusCodes` `[static]`

Map with the strings associated to the most commonly used status codes.

In order to use a new user defined status code, it is necessary to include in this map the associated string.

6.55.4.4 `Response::StatusCodesInitializer http::Response::statusCodesInitializer` `[static], [private]`

The initializer of the `StatusCodes` member.

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

- [http/response.h](#)
- [http/response.cc](#)

6.56 data::Serializer< T > Struct Template Reference

This template class allows to define a "serializer".

```
#include <serialize.h>
```

Collaboration diagram for `data::Serializer< T >`:

Static Public Member Functions

- static [InputStream](#) & [Load](#) ([InputStream](#) &stream, T &var)
- static [OutputStream](#) & [Save](#) ([OutputStream](#) &stream, T &var)

6.56.1 Detailed Description

```
template<typename T>struct data::Serializer< T >
```

This template class allows to define a "serializer".

By default, the basic serializer calls the method `SerializeWith` of the objet to be serialized.

In order to define a serializer of any other specific type, it is required to define a specialization of this template class, and redefine the methods `Load` and `Save`.

See also

[BaseStream](#)

6.56.2 Member Function Documentation

6.56.2.1 `template<typename T> static InputStream& data::Serializer< T >::Load (InputStream & stream, T & var)` `[inline], [static]`

Here is the caller graph for this function:

6.56.2.2 `template<typename T> static OutputStream& data::Serializer< T >::Save (OutputStream & stream, T & var) [inline],[static]`

Here is the caller graph for this function:

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

- data/[serialize.h](#)

6.57 data::Serializer< bool > Struct Template Reference

[Serializer](#) for the `bool` type.

```
#include <serialize.h>
```

Collaboration diagram for `data::Serializer< bool >`:

Static Public Member Functions

- static [InputStream](#) & [Load](#) ([InputStream](#) &*stream*, `bool` &*var*)
- static [OutputStream](#) & [Save](#) ([OutputStream](#) &*stream*, `bool` &*var*)

6.57.1 Detailed Description

```
template<>struct data::Serializer< bool >
```

[Serializer](#) for the `bool` type.

See also

[Serializer](#)

6.57.2 Member Function Documentation

6.57.2.1 `static InputStream& data::Serializer< bool >::Load (InputStream & stream, bool & var) [inline],[static]`

Here is the call graph for this function:

6.57.2.2 `static OutputStream& data::Serializer< bool >::Save (OutputStream & stream, bool & var) [inline],[static]`

Here is the call graph for this function:

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

- data/[serialize.h](#)

6.58 data::Serializer< int > Struct Template Reference

[Serializer](#) for the `int` type.

```
#include <serialize.h>
```

Collaboration diagram for `data::Serializer< int >`:

Static Public Member Functions

- static [InputStream](#) & [Load](#) ([InputStream](#) &stream, int &var)
- static [OutputStream](#) & [Save](#) ([OutputStream](#) &stream, int &var)

6.58.1 Detailed Description

template<>struct data::Serializer< int >

[Serializer](#) for the `int` type.

See also

[Serializer](#)

6.58.2 Member Function Documentation

6.58.2.1 static `InputStream& data::Serializer< int >::Load (InputStream & stream, int & var)` `[inline]`,
`[static]`

Here is the call graph for this function:

6.58.2.2 static `OutputStream& data::Serializer< int >::Save (OutputStream & stream, int & var)` `[inline]`,
`[static]`

Here is the call graph for this function:

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

- data/[serialize.h](#)

6.59 data::Serializer< multimap< string, int > > Struct Template Reference

[Serializer](#) for the `multimap<string,int>` class.

```
#include <serialize.h>
```

Collaboration diagram for `data::Serializer< multimap< string, int > >`:

Static Public Member Functions

- static [InputStream](#) & [Load](#) ([InputStream](#) &stream, `multimap< string, int >` &var)
- static [OutputStream](#) & [Save](#) ([OutputStream](#) &stream, `multimap< string, int >` &var)

6.59.1 Detailed Description

template<>struct data::Serializer< multimap< string, int > >

[Serializer](#) for the `multimap<string,int>` class.

See also

[Serializer](#)

6.59.2 Member Function Documentation

6.59.2.1 `static InputStream& data::Serializer< multimap< string, int > >::Load (InputStream & stream, multimap< string, int > & var)` `[inline]`, `[static]`

Here is the call graph for this function:

6.59.2.2 `static OutputStream& data::Serializer< multimap< string, int > >::Save (OutputStream & stream, multimap< string, int > & var)` `[inline]`, `[static]`

Here is the call graph for this function:

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

- [data/serialize.h](#)

6.60 data::Serializer< string > Struct Template Reference

[Serializer](#) for the `string` class.

```
#include <serialize.h>
```

Collaboration diagram for `data::Serializer< string >`:

Static Public Member Functions

- static [InputStream](#) & [Load](#) ([InputStream](#) &*stream*, `string` &*var*)
- static [OutputStream](#) & [Save](#) ([OutputStream](#) &*stream*, `string` &*var*)

6.60.1 Detailed Description

```
template<>struct data::Serializer< string >
```

[Serializer](#) for the `string` class.

See also

[Serializer](#)

6.60.2 Member Function Documentation

6.60.2.1 `static InputStream& data::Serializer< string >::Load (InputStream & stream, string & var)` `[inline]`, `[static]`

Here is the call graph for this function:

6.60.2.2 `static OutputStream& data::Serializer< string >::Save (OutputStream & stream, string & var)` `[inline]`, `[static]`

Here is the call graph for this function:

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

- [data/serialize.h](#)

6.61 data::Serializer< uint64_t > Struct Template Reference

[Serializer](#) for the `uint64_t` type.

```
#include <serialize.h>
```

Collaboration diagram for data::Serializer< uint64_t >:

Static Public Member Functions

- static [InputStream](#) & [Load](#) ([InputStream](#) &stream, uint64_t &var)
- static [OutputStream](#) & [Save](#) ([OutputStream](#) &stream, uint64_t &var)

6.61.1 Detailed Description

```
template<>struct data::Serializer< uint64_t >
```

[Serializer](#) for the `uint64_t` type.

See also

[Serializer](#)

6.61.2 Member Function Documentation

6.61.2.1 static `InputStream& data::Serializer< uint64_t >::Load (InputStream & stream, uint64_t & var)`
`[inline], [static]`

Here is the call graph for this function:

6.61.2.2 static `OutputStream& data::Serializer< uint64_t >::Save (OutputStream & stream, uint64_t & var)`
`[inline], [static]`

Here is the call graph for this function:

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

- data/[serialize.h](#)

6.62 data::Serializer< vector< T > > Struct Template Reference

[Serializer](#) for the `vector` class.

```
#include <serialize.h>
```

Collaboration diagram for data::Serializer< vector< T > >:

Static Public Member Functions

- static [InputStream](#) & [Load](#) ([InputStream](#) &stream, vector< T > &var)
- static [OutputStream](#) & [Save](#) ([OutputStream](#) &stream, vector< T > &var)

6.62.1 Detailed Description

template<typename T> struct data::Serializer< vector< T > >

[Serializer](#) for the `vector` class.

See also

[Serializer](#)

6.62.2 Member Function Documentation

6.62.2.1 template<typename T> static **InputStream& data::Serializer< vector< T > >::Load (InputStream & stream, vector< T > & var)** [inline],[static]

Here is the call graph for this function:

6.62.2.2 template<typename T> static **OutputStream& data::Serializer< vector< T > >::Save (OutputStream & stream, vector< T > & var)** [inline],[static]

Here is the call graph for this function:

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

- data/[serialize.h](#)

6.63 net::Socket Class Reference

This class has been designed to work with UNIX sockets in an easy and object oriented way.

```
#include <socket.h>
```

Collaboration diagram for net::Socket:

Public Member Functions

- [Socket](#) ()
Initializes the socket id with an invalid value.
- [Socket](#) (int s)
Initializes the socket id with an integer value.
- [Socket](#) (const [Socket](#) &xs)
Copy constructor.
- [operator int](#) () const
This operator allows to work directly with UNIX socket API.
- bool [IsValid](#) () const
- [Socket](#) & [operator=](#) (int nsid)
Copy assignment.
- bool [OpenInet](#) (int type=SOCK_STREAM)
This method creates a new Internet socket, storing its identifier in the object.
- bool [OpenUnix](#) (int type=SOCK_STREAM)
This method creates a new UNIX socket, storing its identifier in the object.
- bool [ListenAt](#) (const [Address](#) &address, int nstack=10)
Configures the socket for listening incoming connections.

- bool [ConnectTo](#) (const [Address](#) &to_address)
Connects the socket to a server.
- bool [BindTo](#) (const [Address](#) &address)
Binds the socket to the specified address.
- int [Accept](#) ([Address](#) *from_address)
If it is a server socket, it accepts a new connection.
- [Socket](#) & [SetBlockingMode](#) (bool state=true)
Set the blocking mode of the send/receive operations.
- bool [IsBlockingMode](#) ()
- int [Receive](#) (void *buf, int len, bool prevent_block=false)
Receives a number of bytes.
- int [ReceiveFrom](#) ([Address](#) *address, void *buf, int len, bool prevent_block=false)
Receives a number of bytes.
- int [Send](#) (void *buf, int len, bool prevent_block=false)
Sends a number of bytes.
- int [SendTo](#) (const [Address](#) &address, void *buf, int len, bool prevent_block=false)
Sends a number of bytes to a specific address.
- bool [SendDescriptor](#) (const [Address](#) &address, int fd, int aux=0)
Sends a descriptor through the socket.
- bool [IsValid](#) ()
Returns true if the sockets is valid, that is, if after a polling regarding error status is not successful.
- int [WaitForInput](#) (int time_out=-1)
Waits until input data is available (POLLIN).
- int [WaitForOutput](#) (int time_out=-1)
Waits until output data can be sent (POLLOUT).
- bool [SetNoDelay](#) (int val=1)
Configures the parameter TCP_NODELAY of the socket.
- bool [ReceiveDescriptor](#) (int *fd, int *aux=NULL)
Receives a descriptor from a socket.
- void [Close](#) ()
Closes the socket.
- [~Socket](#) ()
The destructor does not closes the socket!.

Protected Attributes

- int [sid](#)
Socket id.

6.63.1 Detailed Description

This class has been designed to work with UNIX sockets in an easy and object oriented way.

6.63.2 Constructor & Destructor Documentation

6.63.2.1 net::Socket::Socket () [inline]

Initializes the socket id with an invalid value.

6.63.2.2 `net::Socket::Socket (int s) [inline]`

Initializes the socket id with an integer value.

6.63.2.3 `net::Socket::Socket (const Socket & xs) [inline]`

Copy constructor.

6.63.2.4 `net::Socket::~~Socket () [inline]`

The destructor does not closes the socket!.

6.63.3 Member Function Documentation**6.63.3.1** `int net::Socket::Accept (Address * from_address) [inline]`

If it is a server socket, it accepts a new connection.

Parameters

<i>from_address</i>	Pointer to store the client address.
---------------------	--------------------------------------

Returns

The integer identifier (file descriptor) of the new socket.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.2 `bool net::Socket::BindTo (const Address & address) [inline]`

Binds the socket to the specified address.

Parameters

<i>address</i>	Address to bind.
----------------	----------------------------------

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.3 `void net::Socket::Close () [inline]`

Closes the socket.

Here is the caller graph for this function:

6.63.3.4 `bool net::Socket::ConnectTo (const Address & to_address) [inline]`

Connects the socket to a server.

Parameters

<i>to_address</i>	Address to connect.
-------------------	-------------------------------------

Returns

`true` if successful.

Here is the call graph for this function:

6.63.3.5 `bool net::Socket::IsBlockingMode ()`

Returns

The blocking mode of the send/receive operations.

6.63.3.6 `bool net::Socket::IsValid () const` `[inline]`

Returns

`true` if the identifier has a valid value.

Here is the caller graph for this function:

6.63.3.7 `bool net::Socket::IsValid ()`

Returns `true` if the sockets is valid, that is, if after a polling regarding error status is not successful.

6.63.3.8 `bool net::Socket::ListenAt (const Address & address, int nstack = 10)` `[inline]`

Configures the socket for listening incoming connections.

Parameters

<i>address</i>	Address used to listen.
<i>nstack</i>	Maximum number of clients in listening stack.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.9 `bool net::Socket::OpenInet (int type = SOCK_STREAM)` `[inline]`

This method creates a new Internet socket, storing its identifier in the object.

Parameters

<i>type</i>	Socket type, <code>SOCK_STREAM</code> by default.
-------------	---

Returns

`true` if successful.

Here is the caller graph for this function:

6.63.3.10 `bool net::Socket::OpenUnix (int type = SOCK_STREAM) [inline]`

This method creates a new UNIX socket, storing its identifier in the object.

Parameters

<i>type</i>	Socket type, SOCK_STREAM by default.
-------------	--

Returns

`true` if successful.

Here is the caller graph for this function:

6.63.3.11 `net::Socket::operator int () const` `[inline]`

This operator allows to work directly with UNIX socket API.

6.63.3.12 `Socket& net::Socket::operator= (int nsid)` `[inline]`

Copy assignment.

6.63.3.13 `int net::Socket::Receive (void * buf, int len, bool prevent_block = false)`

Receives a number of bytes.

This methods allows to prevent blocking, without having into account the default blocking mode established.

Parameters

<i>buf</i>	Buffer where to store the received bytes.
<i>len</i>	Length of the buffer.
<i>prevent_block</i>	true if blocking will be prevented.

Returns

The number of received bytes.

Here is the caller graph for this function:

6.63.3.14 `bool net::Socket::ReceiveDescriptor (int * fd, int * aux = NULL)`

Receives a descriptor from a socket.

Parameters

<i>fd</i>	Variable to store the received descriptor.
<i>aux</i>	Auxiliary information received attached.

Returns

`true` if successful.

Here is the caller graph for this function:

6.63.3.15 `int net::Socket::ReceiveFrom (Address * address, void * buf, int len, bool prevent_block = false)`

Receives a number of bytes.

This methods allows to prevent blocking, without having into account the default blocking mode established.

Parameters

<i>address</i>	Pointer to store the from address.
<i>buf</i>	Buffer where to store the received bytes.
<i>len</i>	Length of the buffer.
<i>prevent_block</i>	true if blocking will be prevented.

Returns

The number of received bytes.

Here is the call graph for this function:

6.63.3.16 `int net::Socket::Send (void * buf, int len, bool prevent_block = false)`

Sends a number of bytes.

This methods allows to prevent blocking, without having into account the default blocking mode established.

Parameters

<i>buf</i>	Buffer with the bytes to sent.
<i>len</i>	Number of bytes to sent.
<i>prevent_block</i>	true if blocking will be prevented.

Returns

The number of sent bytes.

Here is the caller graph for this function:

6.63.3.17 `bool net::Socket::SendDescriptor (const Address & address, int fd, int aux = 0)`

Sends a descriptor through the socket.

Parameters

<i>address</i>	Address of the socket to send the descriptor.
<i>fd</i>	File descriptor.
<i>aux</i>	Auxiliary information to send attached.

Returns

true if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.18 `int net::Socket::SendTo (const Address & address, void * buf, int len, bool prevent_block = false)`

Sends a number of bytes to a specific address.

This methods allows to prevent blocking, without having into account the default blocking mode established.

Parameters

<i>address</i>	Address to send the bytes.
<i>buf</i>	Buffer with the bytes to send.
<i>len</i>	Number of bytes to send.
<i>prevent_block</i>	true if blocking will be prevented.

Returns

The number of sent bytes.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.19 Socket & net::Socket::SetBlockingMode (bool *state* = true)

Set the blocking mode of the send/receive operations.

By default, this mode is true.

Parameters

<i>state</i>	Blocking mode state to set.
--------------	-----------------------------

Returns

A reference of the same object.

6.63.3.20 bool net::Socket::SetNoDelay (int *val* = 1) [inline]

Configures the parameter `TCP_NODELAY` of the socket.

Parameters

<i>val</i>	New value for the parameter (1 by default).
------------	---

Returns

true if successful.

6.63.3.21 int net::Socket::WaitForInput (int *time_out* = -1)

Waits until input data is available (`POLLIN`).

Parameters

<i>time_out</i>	Time out (infinite by default).
-----------------	---------------------------------

Returns

The value returned by the `poll` function.

6.63.3.22 int net::Socket::WaitForOutput (int *time_out* = -1)

Waits until output data can be sent (`POLLOUT`).

Parameters

<i>time_out</i>	Time out (infinite by default).
-----------------	---------------------------------

Returns

The value returned by the `poll` function.

6.63.4 Member Data Documentation

6.63.4.1 `int net::Socket::sid` `[protected]`

[Socket](#) id.

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

- [net/socket.h](#)
- [net/socket.cc](#)

6.64 `net::SocketBuffer` Class Reference

Class derived from the STL `std::streambuf` to allow streaming with sockets.

```
#include <socket_stream.h>
```

Inheritance diagram for `net::SocketBuffer`:

Collaboration diagram for `net::SocketBuffer`:

Public Types

- enum { `INPUT_BUFFER_LENGTH` = 500, `OUTPUT_BUFFER_LENGTH` = 500 }

Public Member Functions

- [SocketBuffer](#) (int sid, int `in_len`=`INPUT_BUFFER_LENGTH`, int `out_len`=`OUTPUT_BUFFER_LENGTH`)
- virtual int [sync](#) ()
- virtual int_type [underflow](#) ()
- virtual int_type [overflow](#) (int_type c=EOF)
- int [GetReadBytes](#) () const
- [Socket](#) * [GetSocket](#) ()
- virtual [~SocketBuffer](#) ()

Protected Attributes

- int [sum](#)
- int [in_len](#)
- int [out_len](#)
- char * [in_buf](#)
- char * [out_buf](#)
- [Socket](#) [socket](#)

6.64.1 Detailed Description

Class derived from the STL `std::streambuf` to allow streaming with sockets.

See the documentation related to this STL base class to understand the behaviour of the class [SocketBuffer](#).

See also

`std::streambuf`
[Socket](#)

6.64.2 Member Enumeration Documentation

6.64.2.1 anonymous enum

Enumerator

INPUT_BUFFER_LENGTH
OUTPUT_BUFFER_LENGTH

6.64.3 Constructor & Destructor Documentation

6.64.3.1 `net::SocketBuffer::SocketBuffer (int sid, int in_len = INPUT_BUFFER_LENGTH, int out_len = OUTPUT_BUFFER_LENGTH) [inline]`

6.64.3.2 `virtual net::SocketBuffer::~~SocketBuffer () [inline],[virtual]`

6.64.4 Member Function Documentation

6.64.4.1 `int net::SocketBuffer::GetReadBytes () const [inline]`

6.64.4.2 `Socket* net::SocketBuffer::GetSocket () [inline]`

6.64.4.3 `virtual int_type net::SocketBuffer::overflow (int_type c = EOF) [inline],[virtual]`

Here is the call graph for this function:

6.64.4.4 `virtual int net::SocketBuffer::sync () [inline],[virtual]`

Here is the call graph for this function:

Here is the caller graph for this function:

6.64.4.5 `virtual int_type net::SocketBuffer::underflow () [inline],[virtual]`

Here is the call graph for this function:

6.64.5 Member Data Documentation

6.64.5.1 `char* net::SocketBuffer::in_buf [protected]`

6.64.5.2 `int net::SocketBuffer::in_len [protected]`

6.64.5.3 `char* net::SocketBuffer::out_buf [protected]`

6.64.5.4 `int net::SocketBuffer::out_len` [protected]

6.64.5.5 `Socket net::SocketBuffer::socket` [protected]

6.64.5.6 `int net::SocketBuffer::sum` [protected]

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

- [net/socket_stream.h](#)

6.65 net::SocketStream Class Reference

Class derived from `std::iostream` and [SocketBuffer](#) that represents a socket stream.

```
#include <socket_stream.h>
```

Inheritance diagram for `net::SocketStream`:

Collaboration diagram for `net::SocketStream`:

Public Member Functions

- [SocketStream](#) (int sid, int in_len=INPUT_BUFFER_LENGTH, int out_len=OUTPUT_BUFFER_LENGTH)
- [Socket * operator->](#) ()
- virtual [~SocketStream](#) ()

Additional Inherited Members

6.65.1 Detailed Description

Class derived from `std::iostream` and [SocketBuffer](#) that represents a socket stream.

See also

[std::iostream](#)
[SocketBuffer](#)

6.65.2 Constructor & Destructor Documentation

6.65.2.1 `net::SocketStream::SocketStream (int sid, int in_len = INPUT_BUFFER_LENGTH, int out_len = OUTPUT_BUFFER_LENGTH)` [inline]

6.65.2.2 `virtual net::SocketStream::~~SocketStream ()` [inline], [virtual]

6.65.3 Member Function Documentation

6.65.3.1 `Socket* net::SocketStream::operator-> ()` [inline]

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

- [net/socket_stream.h](#)

6.66 `http::Response::StatusCodesInitializer` Class Reference

Class used for the initializer.

Collaboration diagram for `http::Response::StatusCodesInitializer`:

Public Member Functions

- [StatusCodesInitializer](#) ()

6.66.1 Detailed Description

Class used for the initializer.

6.66.2 Constructor & Destructor Documentation

6.66.2.1 `http::Response::StatusCodesInitializer::StatusCodesInitializer` ()

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

- [http/response.h](#)
- [http/response.cc](#)

6.67 `TraceSystem` Class Reference

Wrapper used by the application to handle the log/trace messages by means of the log4cpp library.

```
#include <trace.h>
```

Collaboration diagram for `TraceSystem`:

Static Public Member Functions

- static bool [AppendToFile](#) (const char *name)
- static bool [AppendToFile](#) (const std::string &name)
- static log4cpp::CategoryStream [logStream](#) ()
- static log4cpp::CategoryStream [errorStream](#) ()
- static log4cpp::CategoryStream [traceStream](#) ()

Private Member Functions

- [TraceSystem](#) ()
- virtual [~TraceSystem](#) ()
- bool [AppendToFile_](#) (const char *name)

Private Attributes

- log4cpp::Category * [category](#)
- log4cpp::Appender * [appender](#)
- log4cpp::PatternLayout * [layout](#)
- log4cpp::Appender * [file_appender](#)
- log4cpp::PatternLayout * [file_layout](#)

Static Private Attributes

- static [TraceSystem](#) `traceSystem`

6.67.1 Detailed Description

Wrapper used by the application to handle the log/trace messages by means of the log4cpp library.

6.67.2 Constructor & Destructor Documentation

6.67.2.1 `TraceSystem::TraceSystem ()` `[private]`

6.67.2.2 `TraceSystem::~~TraceSystem ()` `[private], [virtual]`

6.67.3 Member Function Documentation

6.67.3.1 `static bool TraceSystem::AppendToFile (const char * name)` `[inline], [static]`

Here is the call graph for this function:

Here is the caller graph for this function:

6.67.3.2 `static bool TraceSystem::AppendToFile (const std::string & name)` `[inline], [static]`

Here is the call graph for this function:

6.67.3.3 `bool TraceSystem::AppendToFile_ (const char * name)` `[private]`

Here is the caller graph for this function:

6.67.3.4 `static log4cpp::CategoryStream TraceSystem::errorStream ()` `[inline], [static]`

6.67.3.5 `static log4cpp::CategoryStream TraceSystem::logStream ()` `[inline], [static]`

6.67.3.6 `static log4cpp::CategoryStream TraceSystem::traceStream ()` `[inline], [static]`

6.67.4 Member Data Documentation

6.67.4.1 `log4cpp::Appender* TraceSystem::appender` `[private]`

6.67.4.2 `log4cpp::Category* TraceSystem::category` `[private]`

6.67.4.3 `log4cpp::Appender* TraceSystem::file_appender` `[private]`

6.67.4.4 `log4cpp::PatternLayout* TraceSystem::file_layout` `[private]`

6.67.4.5 `log4cpp::PatternLayout* TraceSystem::layout` `[private]`

6.67.4.6 `TraceSystem TraceSystem::traceSystem` `[static], [private]`

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

- [trace.h](#)
- [trace.cc](#)

6.68 ui Struct Reference

Collaboration diagram for ui:

Static Public Member Functions

- `template<typename T >`
`static void read (T &v, T mn, T mx)`

6.68.1 Member Function Documentation

6.68.1.1 `template<typename T > static void ui::read (T & v, T mn, T mx) [inline], [static]`

Here is the caller graph for this function:

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

- [packet_information.cc](#)

6.69 net::UnixAddress Class Reference

Class to identify and handle an UNIX address.

```
#include <address.h>
```

Inheritance diagram for net::UnixAddress:

Collaboration diagram for net::UnixAddress:

Public Member Functions

- `UnixAddress ()`
Initializes the address to zero.
- `UnixAddress (const UnixAddress &address)`
Copy constructor.
- `UnixAddress (const char *path)`
Initializes the address with given path.
- `UnixAddress & operator= (const UnixAddress &address)`
Copy assignment.
- `UnixAddress & Reset ()`
Removes the file associated to the UNIX address.
- `virtual sockaddr * GetSockAddr () const`
Overloaded from the base class to use the internal address structure.
- `virtual int GetSize () const`
Overloaded from the base class to use the internal address structure.
- `string GetPath () const`
Returns the address path.

Private Attributes

- `sockaddr_un sock_addr`
Internal address structure.

6.69.1 Detailed Description

Class to identify and handle an UNIX address.

The used internal address structure is `sockaddr_un`.

See also

[Address](#)

6.69.2 Constructor & Destructor Documentation

6.69.2.1 `net::UnixAddress::UnixAddress ()` `[inline]`

Initializes the address to zero.

6.69.2.2 `net::UnixAddress::UnixAddress (const UnixAddress & address)` `[inline]`

Copy constructor.

6.69.2.3 `net::UnixAddress::UnixAddress (const char * path)` `[inline]`

Initializes the address with given path.

Parameters

<i>path</i>	Address path.
-------------	-------------------------------

6.69.3 Member Function Documentation

6.69.3.1 `string net::UnixAddress::GetPath () const` `[inline]`

Returns the address path.

6.69.3.2 `virtual int net::UnixAddress::GetSize () const` `[inline], [virtual]`

Overloaded from the base class to use the internal address structure.

Implements [net::Address](#).

6.69.3.3 `virtual sockaddr* net::UnixAddress::GetSockAddr () const` `[inline], [virtual]`

Overloaded from the base class to use the internal address structure.

Implements [net::Address](#).

6.69.3.4 `UnixAddress& net::UnixAddress::operator= (const UnixAddress & address)` `[inline]`

Copy assignment.

6.69.3.5 `UnixAddress& net::UnixAddress::Reset ()` `[inline]`

Removes the file associated to the UNIX address.

Here is the caller graph for this function:

6.69.4 Member Data Documentation

6.69.4.1 sockaddr_un net::UnixAddress::sock_addr [private]

Internal address structure.

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

- [net/address.h](#)

6.70 data::UnlockedAccess Struct Reference

Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_exts.h`.

```
#include <file.h>
```

Collaboration diagram for data::UnlockedAccess:

Static Public Member Functions

- static void [configure](#) (`FILE *file_ptr`)
- static `size_t` [fwrite](#) (`const void *ptr`, `size_t size`, `size_t count`, `FILE *file_ptr`)
- static `size_t` [fread](#) (`void *ptr`, `size_t size`, `size_t count`, `FILE *file_ptr`)
- static `int` [fgetc](#) (`FILE *file_ptr`)
- static `int` [fputc](#) (`int c`, `FILE *file_ptr`)

6.70.1 Detailed Description

Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_exts.h`.

See also

[File](#)

6.70.2 Member Function Documentation

6.70.2.1 static void data::UnlockedAccess::configure (`FILE * file_ptr`) [inline],[static]

6.70.2.2 static int data::UnlockedAccess::fgetc (`FILE * file_ptr`) [inline],[static]

6.70.2.3 static int data::UnlockedAccess::fputc (`int c`, `FILE * file_ptr`) [inline],[static]

6.70.2.4 static `size_t` data::UnlockedAccess::fread (`void * ptr`, `size_t size`, `size_t count`, `FILE * file_ptr`) [inline],[static]

6.70.2.5 static `size_t` data::UnlockedAccess::fwrite (`const void * ptr`, `size_t size`, `size_t count`, `FILE * file_ptr`) [inline],[static]

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

- [data/file.h](#)

6.71 data::vint_vector Class Reference

This class has been implemented with the same philosophy that the class STL vector, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

```
#include <vint_vector.h>
```

Collaboration diagram for data::vint_vector:

Public Member Functions

- [vint_vector](#) ()
Initializes the vector to store 64-bit integers.
- [vint_vector](#) (int [num_bytes](#))
Initializes the vector to store integers with the number of bytes given as parameter.
- [vint_vector](#) (const [vint_vector](#) &v)
Copy constructor.
- const [vint_vector](#) & [operator=](#) (const [vint_vector](#) &v)
Copy assignment.
- void [set_num_bytes](#) (int [num_bytes](#))
Changes the number of bytes of the integer values.
- int [num_bytes](#) () const
Returns the number of bytes used.
- int [data_bytes](#) () const
Returns the current number of bytes stored.
- uint64_t [operator\[\]](#) (int index) const
Operator overloading for indexing the integer values.
- void [push_back](#) (uint64_t value)
Adds a new item to the end of the vector.
- void [clear](#) ()
Clears the content.
- int [size](#) () const
Returns the size of the vector, in number of items.
- uint64_t & [back](#) ()
Return the reference of the last item of the vector.
- virtual [~vint_vector](#) ()

Private Attributes

- uint64_t [mask](#)
Mask used for accessing the data.
- int8_t [num_bytes_](#)
Number of bytes used for the integers.
- vector< uint8_t > [data](#)

6.71.1 Detailed Description

This class has been implemented with the same philosophy that the class STL vector, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

integers of 3 bytes). This class internally handles a vector of 1-byte integers.

See also

[vector](#)

6.71.2 Constructor & Destructor Documentation

6.71.2.1 data::vint_vector::vint_vector () [inline]

Initializes the vector to store 64-bit integers.

6.71.2.2 data::vint_vector::vint_vector (int *num_bytes*) [inline]

Initializes the vector to store integers with the number of bytes given as parameter.

Parameters

<i>num_bytes</i>	Number of bytes of each integer.
------------------	----------------------------------

6.71.2.3 data::vint_vector::vint_vector (const vint_vector & v) [inline]

Copy constructor.

6.71.2.4 virtual data::vint_vector::~~vint_vector () [inline],[virtual]

6.71.3 Member Function Documentation

6.71.3.1 uint64_t& data::vint_vector::back () [inline]

Return the reference of the last item of the vector.

Here is the caller graph for this function:

6.71.3.2 void data::vint_vector::clear () [inline]

Clears the content.

Here is the caller graph for this function:

6.71.3.3 int data::vint_vector::data_bytes () const [inline]

Returns the current number of bytes stored.

6.71.3.4 int data::vint_vector::num_bytes () const [inline]

Returns the number of bytes used.

6.71.3.5 const vint_vector& data::vint_vector::operator= (const vint_vector & v) [inline]

Copy assignment.

6.71.3.6 uint64_t data::vint_vector::operator[] (int *index*) const [inline]

Operator overloading for indexing the integer values.

Parameters

<i>index</i>	Index of the item to return.
--------------	------------------------------

Returns

Value of the item, always as a `uint64_t`.

6.71.3.7 `void data::vint_vector::push_back (uint64_t value) [inline]`

Adds a new item to the end of the vector.

Parameters

<i>value</i>	Value to add to the vector.
--------------	-----------------------------

Here is the caller graph for this function:

6.71.3.8 `void data::vint_vector::set_num_bytes (int num_bytes) [inline]`

Changes the number of bytes of the integer values.

All the current content is removed.

Parameters

<i>num_bytes</i>	New number of bytes to use.
------------------	-----------------------------

Here is the caller graph for this function:

6.71.3.9 `int data::vint_vector::size () const [inline]`

Returns the size of the vector, in number of items.

Here is the caller graph for this function:

6.71.4 Member Data Documentation

6.71.4.1 `vector<uint8_t> data::vint_vector::data [private]`

6.71.4.2 `uint64_t data::vint_vector::mask [private]`

Mask used for accessing the data.

6.71.4.3 `int8_t data::vint_vector::num_bytes_ [private]`

Number of bytes used for the integers.

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

- [data/vint_vector.h](#)

6.72 jpip::WOI Class Reference

Class that identifies a [WOI](#) (Window Of Interest).

```
#include <woi.h>
```

Collaboration diagram for `jpip::WOI`:

Public Member Functions

- [WOI](#) ()
Initializes the resolution level to zero.
- [WOI](#) (const [Point](#) &[position](#), const [Size](#) &[size](#), int [resolution](#))
Initializes the object.
- [WOI](#) (const [WOI](#) &[woi](#))
Copy constructor.
- [WOI](#) & [operator=](#) (const [WOI](#) &[woi](#))
Copy assignment.
- virtual [~WOI](#) ()

Public Attributes

- [Size](#) [size](#)
Size of the [WOI](#) (width and height)
- [Point](#) [position](#)
Position of the upper-left corner of the [WOI](#).
- int [resolution](#)
Resolution level where the [WOI](#) is located (0 == the highest)

Friends

- bool [operator==](#) (const [WOI](#) &[a](#), const [WOI](#) &[b](#))
Returns `true` if the two given [WOIs](#) are equal.
- bool [operator!=](#) (const [WOI](#) &[a](#), const [WOI](#) &[b](#))
Returns `true` if the two given [WOIs](#) are not equal.
- ostream & [operator<<](#) (ostream &[out](#), const [WOI](#) &[woi](#))

6.72.1 Detailed Description

Class that identifies a [WOI](#) (Window Of Interest).

This term refers, from the point of view of the JPIP protocol, to a rectangular region of an image, for a resolution level. This class can be printed.

See also

[Point](#)

6.72.2 Constructor & Destructor Documentation

6.72.2.1 `jpip::WOI::WOI ()` `[inline]`

Initializes the resolution level to zero.

6.72.2.2 `jpip::WOI::WOI (const Point & position, const Size & size, int resolution)` `[inline]`

Initializes the object.

Parameters

<i>position</i>	Position of the WOI .
<i>size</i>	Size of the WOI .
<i>resolution</i>	Resolution level of the WOI .

6.72.2.3 `jpeg::WOI::WOI (const WOI & woi) [inline]`

Copy constructor.

6.72.2.4 `virtual jpeg::WOI::~~WOI () [inline],[virtual]`

6.72.3 Member Function Documentation

6.72.3.1 `WOI& jpeg::WOI::operator= (const WOI & woi) [inline]`

Copy assignment.

6.72.4 Friends And Related Function Documentation

6.72.4.1 `bool operator!= (const WOI & a, const WOI & b) [friend]`

Returns `true` if the two given WOIs are not equal.

6.72.4.2 `ostream& operator<< (ostream & out, const WOI & woi) [friend]`

6.72.4.3 `bool operator== (const WOI & a, const WOI & b) [friend]`

Returns `true` if the two given WOIs are equal.

6.72.5 Member Data Documentation

6.72.5.1 `Point jpeg::WOI::position`

Position of the upper-left corner of the [WOI](#).

6.72.5.2 `int jpeg::WOI::resolution`

Resolution level where the [WOI](#) is located (0 == the highest)

6.72.5.3 `Size jpeg::WOI::size`

Size of the [WOI](#) (width and height)

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

- [jpeg/woi.h](#)

6.73 jpip::WOIComposer Class Reference

By means of this class it is possible to find out the which packets of an image are associated to a [WOI](#).

```
#include <woi_composer.h>
```

Collaboration diagram for jpip::WOIComposer:

Public Member Functions

- [WOIComposer](#) ()
Initializes the object.
- [WOIComposer](#) (const [WOIComposer](#) &composer)
Copy constructor.
- [WOIComposer](#) ([CodingParameters::Ptr](#) coding_parameters)
Initializes the object.
- void [Reset](#) (const [WOI](#) &woi, [CodingParameters::Ptr](#) coding_parameters)
Resets the packets navigation and starts a new one.
- [WOIComposer](#) & [operator=](#) (const [WOIComposer](#) &composer)
Copy assignment.
- [Packet](#) [GetCurrentPacket](#) () const
Returns the current packet.
- bool [GetNextPacket](#) ([Packet](#) *packet=NULL)
Moves to the next packet of the [WOI](#).
- virtual [~WOIComposer](#) ()

Private Attributes

- [Point](#) pxy1
Upper-left corner of the [WOI](#).
- [Point](#) pxy2
Bottom-right corner of the [WOI](#).
- bool [more_packets](#)
Flag to control the last packet.
- int [max_resolution](#)
Maximum resolution.
- [Size](#) [min_precinct_xy](#)
Minimum precinct.
- [Size](#) [max_precinct_xy](#)
Maximum precinct.
- [Packet](#) [current_packet](#)
Current packet.
- [CodingParameters::Ptr](#) [coding_parameters](#)
Pointer to the associated coding parameters.

6.73.1 Detailed Description

By means of this class it is possible to find out the which packets of an image are associated to a [WOI](#).

Given a [WOI](#) and the coding parameters of an image, the code of this class allows to navigate, following the LRCP order, through all the associated packets.

See also

[WOI](#)

CodingParameters

6.73.2 Constructor & Destructor Documentation

6.73.2.1 `jpeg::WOIComposer::WOIComposer ()` [inline]

Initializes the object.

No packets are available.

6.73.2.2 `jpeg::WOIComposer::WOIComposer (const WOIComposer & composer)` [inline]

Copy constructor.

6.73.2.3 `jpeg::WOIComposer::WOIComposer (CodingParameters::Ptr coding_parameters)` [inline]

Initializes the object.

No packets are available.

Parameters

<code>coding_↔ parameters</code>	Pointer to the coding parameters.
--------------------------------------	-----------------------------------

6.73.2.4 `virtual jpeg::WOIComposer::~~WOIComposer ()` [inline],[virtual]

6.73.3 Member Function Documentation

6.73.3.1 `Packet jpeg::WOIComposer::GetCurrentPacket () const` [inline]

Returns the current packet.

Here is the caller graph for this function:

6.73.3.2 `bool jpeg::WOIComposer::GetNextPacket (Packet * packet = NULL)` [inline]

Moves to the next packet of the [WOI](#).

Parameters

<code>packet</code>	Pointer to store the current packet (not the next one).
---------------------	---

Returns

`true` if successful.

Here is the caller graph for this function:

6.73.3.3 `WOIComposer& jpeg::WOIComposer::operator= (const WOIComposer & composer)` [inline]

Copy assignment.

6.73.3.4 `void jpip::WOIComposer::Reset (const WOI & woi, CodingParameters::Ptr coding_parameters) [inline]`

Resets the packets navigation and starts a new one.

Sets the current packet to the first packet of the [WOI](#), assuming a LRCP order.

Parameters

<i>woi</i>	New WOI to use.
<i>coding_↔ parameters</i>	Coding parameters to use.

Here is the caller graph for this function:

6.73.4 Member Data Documentation

6.73.4.1 `CodingParameters::Ptr jpip::WOIComposer::coding_parameters [private]`

Pointer to the associated coding parameters.

6.73.4.2 `Packet jpip::WOIComposer::current_packet [private]`

Current packet.

6.73.4.3 `Size jpip::WOIComposer::max_precinct_xy [private]`

Maximum precinct.

6.73.4.4 `int jpip::WOIComposer::max_resolution [private]`

Maximum resolution.

6.73.4.5 `Size jpip::WOIComposer::min_precinct_xy [private]`

Minimum precinct.

6.73.4.6 `bool jpip::WOIComposer::more_packets [private]`

Flag to control the last packet.

6.73.4.7 `Point jpip::WOIComposer::pxy1 [private]`

Upper-left corner of the [WOI](#).

6.73.4.8 `Point jpip::WOIComposer::pxy2 [private]`

Bottom-right corner of the [WOI](#).

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

- [jpip/woi_composer.h](#)

Chapter 7

File Documentation

7.1 app_config.cc File Reference

```
#include <libconfig.h++>
#include "app_config.h"
Include dependency graph for app_config.cc:
```

7.2 app_config.h File Reference

```
#include <string>
#include <iostream>
Include dependency graph for app_config.h: This graph shows which files directly or indirectly include this file:
```

Classes

- class [AppConfig](#)
Contains the configuration parameters of the application.

7.3 app_info.cc File Reference

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <sys/file.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <sys/sysctl.h>
#include "app_info.h"
Include dependency graph for app_info.cc:
```

Macros

- `#define LOCK_FILE "/tmp/esa_jpip_server.lock"`

7.3.1 Macro Definition Documentation

7.3.1.1 `#define LOCK_FILE "/tmp/esa_jpip_server.lock"`

7.4 app_info.h File Reference

```
#include <assert.h>
#include <iostream>
#include <iomanip>
#include <proc/readproc.h>
```

Include dependency graph for app_info.h: This graph shows which files directly or indirectly include this file:

Classes

- class [AppInfo](#)
Contains the run-time information of the application.
- struct [AppInfo::Data](#)
Contains the data block that is maintained in shared memory.

7.5 args_parser.cc File Reference

```
#include <string>
#include <iomanip>
#include <iostream>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <dirent.h>
#include "trace.h"
#include "args_parser.h"
```

Include dependency graph for args_parser.cc:

7.6 args_parser.h File Reference

```
#include "app_info.h"
#include "app_config.h"
```

Include dependency graph for args_parser.h: This graph shows which files directly or indirectly include this file:

Classes

- class [ArgsParser](#)
Class that allows to parse and handle the application command line parameters.

7.7 base.cc File Reference

```
#include "base.h"
```

Include dependency graph for base.cc:

7.8 base.h File Reference

```
#include <map>
#include <vector>
#include <sstream>
```

Include dependency graph for base.h: This graph shows which files directly or indirectly include this file:

Classes

- struct [base](#)

Contains a set of useful static methods used by the application.

7.9 client_info.cc File Reference

```
#include "client_info.h"
```

Include dependency graph for client_info.cc:

7.10 client_info.h File Reference

```
#include <time.h>
```

Include dependency graph for client_info.h: This graph shows which files directly or indirectly include this file:

Classes

- class [ClientInfo](#)

Contains information of a connected client.

7.11 client_manager.cc File Reference

```
#include "trace.h"
#include "client_manager.h"
#include "jpip/jpip.h"
#include "jpip/request.h"
#include "jpip/databin_server.h"
#include "http/header.h"
#include "http/response.h"
#include "net/socket_stream.h"
#include "jpeg2000/index_manager.h"
```

Include dependency graph for client_manager.cc:

7.12 client_manager.h File Reference

```
#include "app_info.h"
#include "app_config.h"
#include "client_info.h"
#include "jpeg2000/index_manager.h"
```

Include dependency graph for client_manager.h: This graph shows which files directly or indirectly include this file:

Classes

- class [ClientManager](#)

Handles a client connection with a dedicated thread.

7.13 data/data.h File Reference

Namespaces

- [data](#)

Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.14 data/file.cc File Reference

```
#include "file.h"
```

Include dependency graph for file.cc:

7.15 data/file.h File Reference

```
#include <stdio.h>
#include <assert.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string>
#include <stdio_ext.h>
#include "trl_compat.h"
```

Include dependency graph for file.h: This graph shows which files directly or indirectly include this file:

Classes

- struct [data::LockedAccess](#)

Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.

- struct [data::UnlockedAccess](#)

Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_ext.h`.

- class [data::BaseFile](#)< [IO](#) >

This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.

Namespaces

- [data](#)

Contains a set of classes to easy the handling of data and files, as well as the serialization.

Typedefs

- typedef BaseFile< LockedAccess > [data::File](#)

Specialization of the class [BaseFile](#) with locked access.

- typedef BaseFile< UnlockedAccess > [data::FastFile](#)

Specialization of the class [BaseFile](#) with unlocked access.

7.16 data/file_segment.cc File Reference

```
#include "file_segment.h"
```

Include dependency graph for file_segment.cc:

Namespaces

- [data](#)

Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.17 data/file_segment.h File Reference

```
#include <iostream>
```

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

```
#include <assert.h>
```

Include dependency graph for file_segment.h: This graph shows which files directly or indirectly include this file:

Classes

- class [data::FileSegment](#)

Identifies a data segment of a file.

Namespaces

- [data](#)

Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.18 data/serialize.cc File Reference

```
#include "serialize.h"
```

Include dependency graph for serialize.cc:

7.19 data/serialize.h File Reference

```
#include <stdint.h>
#include <string>
#include <vector>
#include <map>
#include "file.h"
```

Include dependency graph for `serialize.h`: This graph shows which files directly or indirectly include this file:

Classes

- struct [data::InputOperator](#)
This struct identifies a basic input operator to be applied to a `File` object.
- struct [data::OutputOperator](#)
This struct identifies a basic output operator to be applied to a `File` object.
- class [data::BaseStream< StreamClass, StreamOperator >](#)
This template is used as the base for the input/output stream classes.
- struct [data::Serializer< T >](#)
This template class allows to define a "serializer".
- class [data::InputStream](#)
Specialization of the `BaseStream` for input serializations.
- class [data::OutputStream](#)
Specialization of the `BaseStream` for output serializations.
- struct [data::Serializer< bool >](#)
Serializer for the `bool` type.
- struct [data::Serializer< int >](#)
Serializer for the `int` type.
- struct [data::Serializer< uint64_t >](#)
Serializer for the `uint64_t` type.
- struct [data::Serializer< string >](#)
Serializer for the `string` class.
- struct [data::Serializer< vector< T > >](#)
Serializer for the `vector` class.
- struct [data::Serializer< multimap< string, int > >](#)
Serializer for the `multimap<string, int>` class.

Namespaces

- [data](#)
Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.20 data/vint_vector.cc File Reference

```
#include "vint_vector.h"
```

Include dependency graph for `vint_vector.cc`:

7.21 data/vint_vector.h File Reference

```
#include <vector>
#include <stdint.h>
#include <assert.h>
#include <algorithm>
```

Include dependency graph for vint_vector.h: This graph shows which files directly or indirectly include this file:

Classes

- class [data::vint_vector](#)

This class has been implemented with the same philosophy that the class STL vector, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

Namespaces

- [data](#)

Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.22 esa_jpip_server.cc File Reference

```
#include <sys/wait.h>
#include <signal.h>
#include "trace.h"
#include "version.h"
#include "app_info.h"
#include "app_config.h"
#include "args_parser.h"
#include "client_info.h"
#include "client_manager.h"
#include "net/poll_table.h"
#include "net/socket_stream.h"
#include "jpeg2000/file_manager.h"
#include "jpeg2000/index_manager.h"
```

Include dependency graph for esa_jpip_server.cc:

Macros

- #define [SERVER_NAME](#) "ESA JPIP Server"
- #define [SERVER_APP_NAME](#) "esa_jpip_server"
- #define [CONFIG_FILE](#) "server.cfg"
- #define [POLLRDHUP](#) (0)

Functions

- int [ChildProcess](#) ()
- void * [ClientThread](#) (void *arg)
- bool [ParseArguments](#) (int argc, char **argv)
- void [SIGCHLD_handler](#) (int signal)
- int [main](#) (int argc, char **argv)

Variables

- [AppConfig cfg](#)
- int `base_id` = 0
- [AppInfo app_info](#)
- [Socket child_socket](#)
- [PollTable poll_table](#)
- bool `child_lost` = false
- [IndexManager index_manager](#)
- [UnixAddress child_address](#) ("/tmp/child_unix_address")
- [UnixAddress father_address](#) ("/tmp/father_unix_address")

7.22.1 Macro Definition Documentation

7.22.1.1 `#define CONFIG_FILE "server.cfg"`

7.22.1.2 `#define POLLRDHUP (0)`

7.22.1.3 `#define SERVER_APP_NAME "esa_jpip_server"`

7.22.1.4 `#define SERVER_NAME "ESA JPIP Server"`

7.22.2 Function Documentation

7.22.2.1 `int ChildProcess ()`

Here is the call graph for this function:

Here is the caller graph for this function:

7.22.2.2 `void * ClientThread (void * arg)`

Here is the call graph for this function:

Here is the caller graph for this function:

7.22.2.3 `int main (int argc, char ** argv)`

Here is the call graph for this function:

7.22.2.4 `bool ParseArguments (int argc, char ** argv)`

7.22.2.5 `void SIGCHLD_handler (int signal)`

Here is the caller graph for this function:

7.22.3 Variable Documentation

7.22.3.1 `AppInfo app_info`

7.22.3.2 `int base_id` = 0

7.22.3.3 `AppConfig cfg`

7.22.3.4 **UnixAddress** child_address("/tmp/child_unix_address")

7.22.3.5 **bool** child_lost = false

7.22.3.6 **Socket** child_socket

7.22.3.7 **UnixAddress** father_address("/tmp/father_unix_address")

7.22.3.8 **IndexManager** index_manager

7.22.3.9 **PollTable** poll_table

7.23 http/header.cc File Reference

```
#include "header.h"
```

Include dependency graph for header.cc:

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.24 http/header.h File Reference

```
#include <string.h>
```

```
#include <iostream>
```

```
#include <assert.h>
```

```
#include "protocol.h"
```

Include dependency graph for header.h: This graph shows which files directly or indirectly include this file:

Classes

- class [http::HeaderName](#)

Container for the strings associated to the most common HTTP headers, used for the specialization of the class [HeaderBase](#).

- class [http::HeaderBase< NAME >](#)

Template class used to identify a HTTP header.

- class [http::HeaderBase< HeaderName::UNDEFINED >](#)

Specialization of the [HeaderBase](#) template class with the [HeaderName : : UNDEFINED](#) value.

- class [http::Header](#)

Class used to handle a HTTP header.

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.25 http/http.h File Reference

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.26 http/protocol.cc File Reference

```
#include "protocol.h"
```

Include dependency graph for protocol.cc:

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.27 http/protocol.h File Reference

```
#include <string>
#include <iostream>
#include <assert.h>
```

Include dependency graph for protocol.h: This graph shows which files directly or indirectly include this file:

Classes

- class [http::Protocol](#)

Class used to identify the HTTP protocol.

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.28 http/request.cc File Reference

```
#include "trace.h"
#include "request.h"
```

Include dependency graph for request.cc:

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

Functions

- istream & [http::operator>>](#) (istream &in, Request &request)
- ostream & [http::operator<<](#) (ostream &out, const Request &request)

7.29 jpip/request.cc File Reference

```
#include "trace.h"
#include "request.h"
Include dependency graph for request.cc:
```

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.30 http/request.h File Reference

```
#include <map>
#include <string>
#include <vector>
#include <sstream>
#include <iostream>
#include <stdlib.h>
#include "header.h"
#include "protocol.h"
```

Include dependency graph for request.h: This graph shows which files directly or indirectly include this file:

Classes

- class [http::Request](#)

Class used to identify a HTTP request (GET or POST).

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.31 jpip/request.h File Reference

```
#include <vector>
#include <string>
#include <iostream>
#include "jpip/woi.h"
#include "jpip/cache_model.h"
#include "http/request.h"
#include "jpeg2000/point.h"
#include "jpeg2000/coding_parameters.h"
#include <string.h>
```

Include dependency graph for request.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpip::Request](#)

Class derived from the HTTP [Request](#) class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.

- union [jpip::Request::ParametersMask](#)

Union used to control the presence of the different JPIP parameters in a request.

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.32 http/response.cc File Reference

```
#include "response.h"
```

Include dependency graph for response.cc:

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.33 http/response.h File Reference

```
#include <map>
#include <string>
#include <sstream>
#include <iostream>
#include "protocol.h"
```

Include dependency graph for response.h: This graph shows which files directly or indirectly include this file:

Classes

- class [http::Response](#)
Class used to identify a HTTP response.
- class [http::Response::StatusCodesInitializer](#)

Class used for the initializer.

Namespaces

- [http](#)

Contains the definition of a set of classes for working easily with the protocol HTTP.

7.34 ipc/event.cc File Reference

```
#include <errno.h>
#include <assert.h>
#include <sys/time.h>
#include "event.h"
```

Include dependency graph for event.cc:

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.35 ipc/event.h File Reference

```
#include <pthread.h>
#include "ipc_object.h"
```

Include dependency graph for event.h: This graph shows which files directly or indirectly include this file:

Classes

- class [ipc::Event](#)

IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the `pthread` mutex and conditional variables API.

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.36 ipc/ipc.h File Reference

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.37 ipc/ipc_object.cc File Reference

```
#include "ipc_object.h"
```

Include dependency graph for ipc_object.cc:

7.38 ipc/ipc_object.h File Reference

```
#include "tr1_compat.h"
```

Include dependency graph for ipc_object.h: This graph shows which files directly or indirectly include this file:

Classes

- class [ipc::IPCObject](#)

Class base of all the IPC classes that has the basic operations (`Init`, `Wait` and `Dispose`) to be overloaded.

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

Enumerations

- enum [ipc::WaitResult](#) { [ipc::WAIT_OBJECT](#) = 0, [ipc::WAIT_TIMEOUT](#), [ipc::WAIT_ERROR](#) }

Enumeration of the possible values returned when a wait operation is performed for an IPC object.

7.39 ipc/mutex.cc File Reference

```
#include <assert.h>
#include <errno.h>
#include <sched.h>
#include "mutex.h"
```

Include dependency graph for mutex.cc:

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.40 ipc/mutex.h File Reference

```
#include <pthread.h>
#include "ipc_object.h"
```

Include dependency graph for mutex.h: This graph shows which files directly or indirectly include this file:

Classes

- class [ipc::Mutex](#)

IPC object that offers the functionality of a mutex, implemented by means of the `pthread` mutex API.

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.41 ipc/rdwr_lock.cc File Reference

```
#include "rdwr_lock.h"
#include <assert.h>
#include <errno.h>
#include <sched.h>
#include <iostream>
```

Include dependency graph for rdwr_lock.cc:

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.42 ipc/rdwr_lock.h File Reference

```
#include <pthread.h>
```

```
#include "ipc_object.h"
```

Include dependency graph for rdwr_lock.h: This graph shows which files directly or indirectly include this file:

Classes

- class [ipc::RdWrLock](#)

IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

Namespaces

- [ipc](#)

Contains classes for working with the IPC mechanisms available in Linux using the pthread library.

7.43 jpeg2000/codestream_index.cc File Reference

```
#include "codestream_index.h"
```

Include dependency graph for codestream_index.cc:

7.44 jpeg2000/codestream_index.h File Reference

```
#include <vector>
```

```
#include "base.h"
```

```
#include "data/file_segment.h"
```

Include dependency graph for codestream_index.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::CodestreamIndex](#)

Class used for indexing the information of a JPEG2000 codestream.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.45 jpeg2000/coding_parameters.cc File Reference

```
#include "coding_parameters.h"
```

Include dependency graph for coding_parameters.cc:

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.46 jpeg2000/coding_parameters.h File Reference

```
#include <vector>
#include <math.h>
#include "trl_compat.h"
#include "base.h"
#include "point.h"
#include "trace.h"
#include "packet.h"
```

Include dependency graph for coding_parameters.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::CodingParameters](#)

Contains the coding parameters of a JPEG2000 image codestream.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.47 jpeg2000/file_manager.cc File Reference

```
#include "file_manager.h"
#include "trace.h"
```

Include dependency graph for file_manager.cc:

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

Macros

- #define [EOC_MARKER](#) 0xFFD9
- #define [SOC_MARKER](#) 0xFF4F
- #define [SIZ_MARKER](#) 0xFF51
- #define [COD_MARKER](#) 0xFF52
- #define [SOT_MARKER](#) 0xFF90
- #define [PLT_MARKER](#) 0xFF58
- #define [SOD_MARKER](#) 0xFF93

- `#define JP2C_BOX_ID 0x6A703263`
- `#define XML_BOX_ID 0x786D6C20`
- `#define ASOC_BOX_ID 0x61736F63`
- `#define NLST_BOX_ID 0x6E6C7374`
- `#define JPCH_BOX_ID 0x6A706368`
- `#define FTBL_BOX_ID 0x6674626C`
- `#define DBTL_BOX_ID 0x6474626C`
- `#define URL_BOX_ID 0x75726C20`
- `#define FLST_BOX_ID 0x666C7374`

7.47.1 Macro Definition Documentation

7.47.1.1 `#define ASOC_BOX_ID 0x61736F63`

7.47.1.2 `#define COD_MARKER 0xFF52`

7.47.1.3 `#define DBTL_BOX_ID 0x6474626C`

7.47.1.4 `#define EOC_MARKER 0xFFD9`

7.47.1.5 `#define FLST_BOX_ID 0x666C7374`

7.47.1.6 `#define FTBL_BOX_ID 0x6674626C`

7.47.1.7 `#define JP2C_BOX_ID 0x6A703263`

7.47.1.8 `#define JPCH_BOX_ID 0x6A706368`

7.47.1.9 `#define NLST_BOX_ID 0x6E6C7374`

7.47.1.10 `#define PLT_MARKER 0xFF58`

7.47.1.11 `#define SIZ_MARKER 0xFF51`

7.47.1.12 `#define SOC_MARKER 0xFF4F`

7.47.1.13 `#define SOD_MARKER 0xFF93`

7.47.1.14 `#define SOT_MARKER 0xFF90`

7.47.1.15 `#define URL_BOX_ID 0x75726C20`

7.47.1.16 `#define XML_BOX_ID 0x786D6C20`

7.48 jpeg2000/file_manager.h File Reference

```
#include <sys/stat.h>
#include "data/serialize.h"
#include "image_info.h"
```

Include dependency graph for file_manager.h: This graph shows which files directly or indirectly include this file:

Classes

- class `jpeg2000::FileManager`

Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.49 jpeg2000/image_index.cc File Reference

```
#include "trace.h"
#include "image_index.h"
Include dependency graph for image_index.cc:
```

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.50 jpeg2000/image_index.h File Reference

```
#include "trace.h"
#include <list>
#include <vector>
#include "base.h"
#include "range.h"
#include "image_info.h"
#include "packet_index.h"
#include "ipc/rdwr_lock.h"
Include dependency graph for image_index.h: This graph shows which files directly or indirectly include this file:
```

Classes

- class [jpeg2000::ImageIndex](#)

Contains the indexing information of a JPEG2000 image file that is managed by the index manager.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.51 jpeg2000/image_info.cc File Reference

```
#include "image_info.h"
Include dependency graph for image_info.cc:
```


7.52 jpeg2000/image_info.h File Reference

```
#include <map>
#include "base.h"
#include "data/file.h"
#include "codestream_index.h"
#include "coding_parameters.h"
#include "meta_data.h"
```

Include dependency graph for image_info.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::ImageInfo](#)
Contains the indexing information of a JPEG2000 image.

Namespaces

- [jpeg2000](#)
Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.53 jpeg2000/index_manager.cc File Reference

```
#include <assert.h>
#include "trace.h"
#include "index_manager.h"
Include dependency graph for index_manager.cc:
```

Namespaces

- [jpeg2000](#)
Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.54 jpeg2000/index_manager.h File Reference

```
#include <list>
#include "ipc/mutex.h"
#include "image_index.h"
#include "file_manager.h"
```

Include dependency graph for index_manager.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::IndexManager](#)
Manages the indexing information of a repository fo images.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.55 jpeg2000/jpeg2000.h File Reference

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.56 jpeg2000/meta_data.cc File Reference

```
#include "meta_data.h"
```

Include dependency graph for meta_data.cc:

7.57 jpeg2000/meta_data.h File Reference

```
#include <vector>
#include "base.h"
#include "jpeg2000/place_holder.h"
```

Include dependency graph for meta_data.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::Metadata](#)

Contains the indexing information associated to the meta-data of a JPEG2000 image file.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.58 jpeg2000/packet.cc File Reference

```
#include "packet.h"
```

Include dependency graph for packet.cc:

7.59 jpeg2000/packet.h File Reference

```
#include "point.h"
```

Include dependency graph for packet.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::Packet](#)
Contains the information of a packet.

Namespaces

- [jpeg2000](#)
Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.60 jpeg2000/packet_index.cc File Reference

```
#include "packet_index.h"
```

Include dependency graph for packet_index.cc:

7.61 jpeg2000/packet_index.h File Reference

```
#include "data/vint_vector.h"
```

```
#include "data/file_segment.h"
```

Include dependency graph for packet_index.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::PacketIndex](#)
Class used for indexing the packets of a codestream image.

Namespaces

- [jpeg2000](#)
Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.62 jpeg2000/place_holder.cc File Reference

```
#include "place_holder.h"
```

Include dependency graph for place_holder.cc:

7.63 jpeg2000/place_holder.h File Reference

```
#include "data/file_segment.h"
```

Include dependency graph for place_holder.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::PlaceHolder](#)
Contains the information of a place-holder.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.64 jpeg2000/point.cc File Reference

```
#include "point.h"
```

Include dependency graph for point.cc:

7.65 jpeg2000/point.h File Reference

```
#include <iostream>
```

Include dependency graph for point.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::Point](#)

Represents a couple of integer values that can be used to identify a coordinate as well as a size.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

Typedefs

- typedef Point [jpeg2000::Size](#)

It is a synonymous of the class [Point](#).

7.66 jpeg2000/range.cc File Reference

```
#include "range.h"
```

Include dependency graph for range.cc:

7.67 jpeg2000/range.h File Reference

```
#include <iostream>
```

```
#include <assert.h>
```

Include dependency graph for range.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpeg2000::Range](#)

Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

Namespaces

- [jpeg2000](#)

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.68 jpip/cache_model.cc File Reference

```
#include "cache_model.h"
```

Include dependency graph for cache_model.cc:

7.69 jpip/cache_model.h File Reference

```
#include <vector>
#include <iostream>
#include <limits.h>
#include "base.h"
#include "jpip/jpip.h"
#include "data/serialize.h"
```

Include dependency graph for cache_model.h: This graph shows which files directly or indirectly include this file:

Classes

- struct [jpip::DataBinSelector< BIN_CLASS >](#)
Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.
- class [jpip::CacheModel](#)
The cache model of a JPIP client is handled using this class.
- class [jpip::CacheModel::Codestream](#)
Sub-class of the cache model class used to identify a codestream.
- struct [jpip::DataBinSelector< DataBinClass::META_DATA >](#)
- struct [jpip::DataBinSelector< DataBinClass::MAIN_HEADER >](#)
- struct [jpip::DataBinSelector< DataBinClass::TILE_HEADER >](#)
- struct [jpip::DataBinSelector< DataBinClass::PRECINCT >](#)

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.70 jpip/databin_server.cc File Reference

```
#include "databin_server.h"
#include "data/file_segment.h"
```

Include dependency graph for databin_server.cc:

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.71 jpip/databin_server.h File Reference

```
#include <utility>
#include "trace.h"
#include "data/file.h"
#include "jpip/woi.h"
#include "jpip/request.h"
#include "jpip/cache_model.h"
#include "jpip/woi_composer.h"
#include "jpip/databin_writer.h"
#include "jpeg2000/range.h"
#include "jpeg2000/image_index.h"
```

Include dependency graph for databin_server.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpip::DataBinServer](#)

Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.72 jpip/databin_writer.cc File Reference

```
#include "databin_writer.h"
```

Include dependency graph for databin_writer.cc:

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.73 jpip/databin_writer.h File Reference

```
#include <stdint.h>
#include "jpip.h"
#include "data/file.h"
#include "data/file_segment.h"
#include "jpeg2000/place_holder.h"
```

Include dependency graph for databin_writer.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpip::DataBinWriter](#)

Class used to generate data-bin segments and write them into a memory buffer.

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.74 jpip/jpip.cc File Reference

```
#include "jpip.h"
```

Include dependency graph for jpip.cc:

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.75 jpip/jpip.h File Reference

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

Classes

- class [jpip::DataBinClass](#)

Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.

- class [jpip::EOR](#)

Class that contains all the definitions of the EOF messages defined for the JPIP protocol.

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.76 jpip/woi.cc File Reference

```
#include "woi.h"
```

Include dependency graph for woi.cc:

7.77 jpip/woi.h File Reference

```
#include <iostream>
```

```
#include "jpeg2000/point.h"
```

Include dependency graph for woi.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpip::WOI](#)

Class that identifies a [WOI](#) (Window Of Interest).

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.78 jpip/woi_composer.cc File Reference

7.79 jpip/woi_composer.h File Reference

```
#include "woi.h"
#include "jpeg2000/packet.h"
#include "jpeg2000/coding_parameters.h"
```

Include dependency graph for woi_composer.h: This graph shows which files directly or indirectly include this file:

Classes

- class [jpip::WOIComposer](#)

By means of this class it is possible to find out the which packets of an image are associated to a [WOI](#).

Namespaces

- [jpip](#)

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.80 net/address.cc File Reference

```
#include "address.h"
```

Include dependency graph for address.cc:

7.81 net/address.h File Reference

```
#include <string>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <sys/un.h>
#include <unistd.h>
```

Include dependency graph for address.h: This graph shows which files directly or indirectly include this file:

Classes

- class [net::Address](#)

Abstract base class to wrap the `sockaddr` derived structures.

- class [net::InetAddress](#)

Class to identify and handle an Internet address.

- class [net::UnixAddress](#)

Class to identify and handle an UNIX address.

Namespaces

- [net](#)

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.82 net/net.h File Reference

Namespaces

- [net](#)

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.83 net/poll_table.cc File Reference

```
#include "poll_table.h"
```

Include dependency graph for poll_table.cc:

7.84 net/poll_table.h File Reference

```
#include <vector>
```

```
#include <poll.h>
```

```
#include <algorithm>
```

Include dependency graph for poll_table.h: This graph shows which files directly or indirectly include this file:

Classes

- struct [net::PollFD](#)

Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.

- class [net::PollTable](#)

This class allows to perform polls easily over a vector of descriptors.

Namespaces

- [net](#)

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.85 net/socket.cc File Reference

```
#include <fcntl.h>
```

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <assert.h>
#include "socket.h"
#include "poll_table.h"
Include dependency graph for socket.cc:
```

Namespaces

- [net](#)

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

Macros

- `#define` [POLLRDHUP](#) (0)

7.85.1 Macro Definition Documentation

7.85.1.1 `#define` POLLRDHUP (0)

7.86 net/socket.h File Reference

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/times.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string>
#include "address.h"
```

Include dependency graph for socket.h: This graph shows which files directly or indirectly include this file:

Classes

- class [net::Socket](#)

This class has been designed to work with UNIX sockets in an easy and object oriented way.

Namespaces

- [net](#)

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.87 net/socket_stream.cc File Reference

```
#include "socket_stream.h"
```

Include dependency graph for socket_stream.cc:

7.88 net/socket_stream.h File Reference

```
#include <iostream>
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include "socket.h"
```

Include dependency graph for socket_stream.h: This graph shows which files directly or indirectly include this file:

Classes

- class [net::SocketBuffer](#)

Class derived from the STL `std::streambuf` to allow streaming with sockets.

- class [net::SocketStream](#)

Class derived from `std::iostream` and [SocketBuffer](#) that represents a socket stream.

Namespaces

- [net](#)

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.89 packet_information.cc File Reference

```
#include "trace.h"
```

```
#include "app_config.h"
```

```
#include "jpip/woi_composer.h"
```

```
#include "jpeg2000/index_manager.h"
```

Include dependency graph for packet_information.cc:

Classes

- struct [ui](#)

Functions

- int [main](#) (void)

7.89.1 Function Documentation

7.89.1.1 int main (void)

Here is the call graph for this function:

7.90 tr1_compat.h File Reference

```
#include <tr1/memory>
```

Include dependency graph for tr1_compat.h: This graph shows which files directly or indirectly include this file:

Macros

- #define [SHARED_PTR](#) std::tr1::shared_ptr

7.90.1 Macro Definition Documentation

7.90.1.1 #define [SHARED_PTR](#) std::tr1::shared_ptr

7.91 trace.cc File Reference

```
#include "trace.h"
```

Include dependency graph for trace.cc:

7.92 trace.h File Reference

```
#include <string>
#include <iostream>
#include <log4cpp/Category.hh>
#include <log4cpp/FileAppender.hh>
#include <log4cpp/PatternLayout.hh>
#include <log4cpp/OstreamAppender.hh>
```

Include dependency graph for trace.h: This graph shows which files directly or indirectly include this file:

Classes

- class [TraceSystem](#)

Wrapper used by the application to handle the log/trace messages by means of the log4cpp library.

Macros

- #define [LOG4CPP_FIX_ERROR_COLLISION](#) 1
- #define [_RED](#) "31m"
- #define [_GREEN](#) "32m"
- #define [_YELLOW](#) "33m"
- #define [_BLUE](#) "34m"
- #define [_SET_COLOR](#)(a) "\033[" a
- #define [_RESET_COLOR](#)() "\033[0m"
- #define [LOG](#)(a) ([TraceSystem::logStream](#)() << a << [log4cpp::eol](#))
- #define [LOGC](#)(c, a) ([TraceSystem::logStream](#)() << [_SET_COLOR](#)(c) << a << [_RESET_COLOR](#)() << [log4cpp::eol](#))
- #define [ERROR](#)(a) ([TraceSystem::errorStream](#)() << [_SET_COLOR](#)([_RED](#)) << [__FILE__](#) << ":" << [__LINE__](#) << ": ERROR: " << a << [_RESET_COLOR](#)() << [log4cpp::eol](#))
- #define [TRACE](#)(a) {}
- #define [CERR](#)(a) (cerr << [_SET_COLOR](#)([_RED](#)) << a << "!" << [_RESET_COLOR](#)() << endl, -1)

7.92.1 Macro Definition Documentation

7.92.1.1 `#define _BLUE "34m"`

7.92.1.2 `#define _GREEN "32m"`

7.92.1.3 `#define _RED "31m"`

7.92.1.4 `#define _RESET_COLOR() "\033[0m"`

7.92.1.5 `#define _SET_COLOR(a) "\033[" a`

7.92.1.6 `#define _YELLOW "33m"`

7.92.1.7 `#define CERR(a) (cerr << _SET_COLOR(_RED) << a << "!" << _RESET_COLOR() << endl, -1)`

7.92.1.8 `#define ERROR(a)(TraceSystem::errorStream() << _SET_COLOR(_RED) << __FILE__ << ":" << __LINE__ << ": ERROR: " << a << _RESET_COLOR() << log4cpp::eol)`

7.92.1.9 `#define LOG(a)(TraceSystem::logStream() << a << log4cpp::eol)`

7.92.1.10 `#define LOG4CPP_FIX_ERROR_COLLISION 1`

7.92.1.11 `#define LOGC(c, a)(TraceSystem::logStream() << _SET_COLOR(c) << a << _RESET_COLOR() << log4cpp::eol)`

7.92.1.12 `#define TRACE(a) {}`

7.93 version.h File Reference

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

Macros

- `#define VERSION "0.2"`

7.93.1 Macro Definition Documentation

7.93.1.1 `#define VERSION "0.2"`

Index

- `_BLUE`
 - `trace.h`, [219](#)
 - `_GREEN`
 - `trace.h`, [219](#)
 - `_RED`
 - `trace.h`, [219](#)
 - `_RESET_COLOR`
 - `trace.h`, [219](#)
 - `_SET_COLOR`
 - `trace.h`, [219](#)
 - `_YELLOW`
 - `trace.h`, [219](#)
 - `~Address`
 - `net::Address`, [17](#)
 - `~AppConfig`
 - `AppConfig`, [19](#)
 - `~AppInfo`
 - `AppInfo`, [24](#)
 - `~BaseFile`
 - `data::BaseFile`, [31](#)
 - `~BaseStream`
 - `data::BaseStream`, [37](#)
 - `~CacheModel`
 - `jipip::CacheModel`, [40](#)
 - `~ClientInfo`
 - `ClientInfo`, [44](#)
 - `~ClientManager`
 - `ClientManager`, [47](#)
 - `~CodestreamIndex`
 - `jpeg2000::CodestreamIndex`, [52](#)
 - `~CodingParameters`
 - `jpeg2000::CodingParameters`, [55](#)
 - `~DataBinServer`
 - `jipip::DataBinServer`, [65](#)
 - `~DataBinWriter`
 - `jipip::DataBinWriter`, [71](#)
 - `~FileManager`
 - `jpeg2000::FileManager`, [83](#)
 - `~FileSegment`
 - `data::FileSegment`, [91](#)
 - `~IPCObject`
 - `ipc::IPCObject`, [117](#)
 - `~ImageIndex`
 - `jpeg2000::ImageIndex`, [101](#)
 - `~ImageInfo`
 - `jpeg2000::ImageInfo`, [107](#)
 - `~IndexManager`
 - `jpeg2000::IndexManager`, [109](#)
 - `~Metadata`
 - `jpeg2000::Metadata`, [120](#)
 - `~Packet`
 - `jpeg2000::Packet`, [126](#)
 - `~PacketIndex`
 - `jpeg2000::PacketIndex`, [128](#)
 - `~Placeholder`
 - `jpeg2000::Placeholder`, [133](#)
 - `~Point`
 - `jpeg2000::Point`, [137](#)
 - `~PollTable`
 - `net::PollTable`, [141](#)
 - `~Range`
 - `jpeg2000::Range`, [145](#)
 - `~Request`
 - `jipip::Request`, [150](#)
 - `~Socket`
 - `net::Socket`, [166](#)
 - `~SocketBuffer`
 - `net::SocketBuffer`, [173](#)
 - `~SocketStream`
 - `net::SocketStream`, [174](#)
 - `~TraceSystem`
 - `TraceSystem`, [176](#)
 - `~WOI`
 - `jipip::WOI`, [184](#)
 - `~WOIComposer`
 - `jipip::WOIComposer`, [186](#)
 - `~vint_vector`
 - `data::vint_vector`, [181](#)
- `ASOC_BOX_ID`
 - `file_manager.cc`, [205](#)
- `Accept`
 - `net::Socket`, [166](#)
- `Add`
 - `jpeg2000::PacketIndex`, [128](#)
 - `net::PollTable`, [141](#)
- `AddTo`
 - `jipip::DataBinSelector< DataBinClass::MAIN_HEADER >`, [62](#)
 - `jipip::DataBinSelector< DataBinClass::META_DATA >`, [62](#)
 - `jipip::DataBinSelector< DataBinClass::PRECINCT >`, [63](#)
 - `jipip::DataBinSelector< DataBinClass::TILE_HEADER >`, [63](#)
- `AddToDataBin`
 - `jipip::CacheModel`, [41](#)
- `AddToMainHeader`
 - `jipip::CacheModel::Codestream`, [49](#)

- AddToMetadata
 - jipip::CacheModel, 41
- AddToPrecinct
 - jipip::CacheModel::Codestream, 49
- AddToTileHeader
 - jipip::CacheModel::Codestream, 49
- Address
 - net::Address, 17
- address
 - AppConfig, 20
- address_
 - AppConfig, 21
- app_config.cc, 189
- app_config.h, 189
- app_info
 - ArgsParser, 27
 - ClientManager, 47
 - esa_jpip_server.cc, 196
- app_info.cc, 189
- LOCK_FILE, 190
- app_info.h, 190
- AppConfig, 18
 - ~AppConfig, 19
 - address, 20
 - address_, 21
 - AppConfig, 19
 - cache_max_time, 20
 - cache_max_time_, 21
 - caching_folder, 20
 - caching_folder_, 21
 - com_time_out, 20
 - com_time_out_, 21
 - images_folder, 20
 - images_folder_, 21
 - Load, 20
 - log_requests, 20
 - log_requests_, 22
 - logging, 20
 - logging_, 22
 - logging_folder, 21
 - logging_folder_, 22
 - max_chunk_size, 21
 - max_chunk_size_, 22
 - max_connections, 21
 - max_connections_, 22
 - operator<<, 21
 - port, 21
 - port_, 22
- AppInfo, 22
 - ~AppInfo, 24
 - AppInfo, 24
 - available_memory, 24
 - available_memory_, 25
 - child_memory, 24
 - child_memory_, 25
 - child_time, 24
 - child_time_, 25
 - data_ptr, 25
 - father_memory, 24
 - father_memory_, 26
 - GetProcStat, 24
 - GetProcStat_, 24
 - Init, 24
 - is_running, 25
 - is_running_, 26
 - lock_file, 26
 - num_threads, 25
 - num_threads_, 26
 - operator<<, 25
 - operator->, 25
 - shmId, 26
 - time, 25
 - time_, 26
 - Update, 25
- AppInfo::Data, 59
 - child_iterations, 60
 - child_pid, 60
 - father_pid, 60
 - num_connections, 60
 - Reset, 60
- AppendToFile
 - TraceSystem, 176
- AppendToFile_
 - TraceSystem, 176
- appender
 - TraceSystem, 176
- args_parser.cc, 190
- args_parser.h, 190
- ArgsParser, 26
 - app_info, 27
 - ArgsParser, 27
 - cfg, 27
 - Parse, 27
- aux
 - jpeg2000::PacketIndex, 130
- available_memory
 - AppInfo, 24
- available_memory_
 - AppInfo, 25
- BYTE_LIMIT_REACHED
 - jipip::EOR, 78
- back
 - data::vint_vector, 181
- base, 28
 - copy, 28
 - to_string, 28
- base.cc, 191
- base.h, 191
- base_id
 - ClientInfo, 44
 - esa_jpip_server.cc, 196
- base_id_
 - ClientInfo, 44
- BaseFile
 - data::BaseFile, 31
- BaseStream

- data::BaseStream, 37
- BindTo
 - net::Socket, 166
- BuildIndex
 - jpeg2000::ImageIndex, 101
- bytes_sent
 - ClientInfo, 44
- bytes_sent_
 - ClientInfo, 44
- CACHE_CONTROL
 - http::HeaderName, 98
- CERR
 - trace.h, 219
- CLOSEST
 - jpip::Request, 150
- COD_MARKER
 - file_manager.cc, 205
- CONFIG_FILE
 - esa_jpip_server.cc, 196
- CONTENT_LENGTH
 - http::HeaderName, 98
- CONTENT_TYPE
 - http::HeaderName, 98
- CPRL_PROGRESSION
 - jpeg2000::CodingParameters, 55
- CRLF
 - http::Protocol, 144
- cache_dir
 - jpeg2000::FileManager, 83
- cache_dir_
 - jpeg2000::FileManager, 89
- cache_max_time
 - AppConfig, 20
- cache_max_time_
 - AppConfig, 21
- cache_model
 - jpip::DataBinServer, 68
 - jpip::Request, 152
- CacheControl
 - http::Header, 93
- CacheModel
 - jpip::CacheModel, 40
- caching_folder
 - AppConfig, 20
- caching_folder_
 - AppConfig, 21
- category
 - TraceSystem, 176
- cclose
 - jpip::Request::ParametersMask, 131
- cfg
 - ArgsParser, 27
 - ClientManager, 47
 - esa_jpip_server.cc, 196
- child_address
 - esa_jpip_server.cc, 196
- child_iterations
 - AppInfo::Data, 60
- child_lost
 - esa_jpip_server.cc, 197
- child_memory
 - AppInfo, 24
- child_memory_
 - AppInfo, 25
- child_pid
 - AppInfo::Data, 60
- child_socket
 - esa_jpip_server.cc, 197
- child_time
 - AppInfo, 24
- child_time_
 - AppInfo, 25
- ChildProcess
 - esa_jpip_server.cc, 196
- cid
 - jpip::Request::ParametersMask, 131
- Clear
 - jpeg2000::CodestreamIndex, 52
 - jpeg2000::PacketIndex, 128
 - jpip::CacheModel, 41
 - jpip::Request::ParametersMask, 131
- clear
 - data::vint_vector, 181
- ClearPreviousIds
 - jpip::DataBinWriter, 71
- client_info.cc, 191
- client_info.h, 191
- client_manager.cc, 191
- client_manager.h, 192
- ClientInfo, 43
 - ~ClientInfo, 44
 - base_id, 44
 - base_id_, 44
 - bytes_sent, 44
 - bytes_sent_, 44
 - ClientInfo, 44
 - father_sock, 44
 - father_sock_, 44
 - sock, 44
 - sock_, 45
 - time, 44
 - tm_start, 45
- ClientManager, 45
 - ~ClientManager, 47
 - app_info, 47
 - cfg, 47
 - ClientManager, 46
 - index_manager, 47
 - Run, 47
 - RunBasic, 47
- ClientThread
 - esa_jpip_server.cc, 196
- Close
 - data::BaseFile, 31
 - data::BaseStream, 38
 - net::Socket, 166

- ClosedImage
 - jpeg2000::IndexManager, 110
- cnew
 - jpeg2000::Request::ParametersMask, 131
- code
 - http::Response, 158
- Codestream
 - jpeg2000::CacheModel::Codestream, 49
- codestream_idx
 - jpeg2000::DataBinWriter, 76
- CodestreamIndex
 - jpeg2000::CodestreamIndex, 52
- codestreams
 - jpeg2000::ImageIndex, 105
 - jpeg2000::ImageInfo, 108
 - jpeg2000::CacheModel, 42
- coding_parameters
 - jpeg2000::ImageIndex, 105
 - jpeg2000::ImageInfo, 108
 - jpeg2000::WOIComposer, 187
- CodingParameters
 - jpeg2000::CodingParameters, 55
- com_time_out
 - AppConfig, 20
- com_time_out_
 - AppConfig, 21
- component
 - jpeg2000::Packet, 126
- condv
 - ipc::Event, 81
- configure
 - data::LockedAccess, 119
 - data::UnlockedAccess, 179
- ConnectTo
 - net::Socket, 166
- ContentLength
 - http::Header, 93
- ContentType
 - http::Header, 93
- context
 - jpeg2000::Request::ParametersMask, 131
- copy
 - base, 28
- current_idx
 - jpeg2000::DataBinServer, 68
- current_packet
 - jpeg2000::WOIComposer, 187
- DBTL_BOX_ID
 - file_manager.cc, 205
- data, 11
 - data::vint_vector, 182
 - FastFile, 12
 - File, 12
- data/data.h, 192
- data/file.cc, 192
- data/file.h, 192
- data/file_segment.cc, 193
- data/file_segment.h, 193
- data/serialize.cc, 193
- data/serialize.h, 194
- data/vint_vector.cc, 194
- data/vint_vector.h, 195
- data::BaseFile
 - ~BaseFile, 31
 - BaseFile, 31
 - Close, 31
 - Exists, 31
 - file_ptr, 36
 - GetDescriptor, 31
 - GetOffset, 31
 - GetSize, 31
 - IsEOF, 31
 - IsValid, 31
 - Open, 31, 33
 - OpenForReading, 33
 - OpenForWriting, 33
 - operator bool, 33
 - Ptr, 30
 - Read, 34
 - ReadByte, 34
 - ReadReverse, 34
 - Seek, 34
 - Write, 34
 - WriteByte, 36
 - WriteReverse, 36
- data::BaseFile< IO >, 29
- data::BaseStream
 - ~BaseStream, 37
 - BaseStream, 37
 - Close, 38
 - file_, 39
 - Open, 38
 - operator bool, 38
 - operator&, 38
 - result, 38
 - result_, 39
 - SerializeBytes, 38
- data::BaseStream< StreamClass, StreamOperator >, 36
- data::FileSegment, 90
 - ~FileSegment, 91
 - FileSegment, 91
 - IsContiguousTo, 91
 - length, 92
 - Null, 92
 - offset, 92
 - operator!=, 91
 - operator<<, 92
 - operator=, 91
 - operator==, 91
 - RemoveFirst, 91
 - RemoveLast, 92
 - SerializeWith, 92
- data::InputOperator, 115
 - FileAccess, 115
 - SerializeBytes, 115

- data::InputStream, 116
 - Serialize, 116
- data::LockedAccess, 119
 - configure, 119
 - fgetc, 119
 - fputc, 119
 - fread, 119
 - fwrite, 119
- data::OutputOperator, 123
 - FileAccess, 124
 - SerializeBytes, 124
- data::OutputStream, 124
 - Serialize, 125
- data::Serializer
 - Load, 159
 - Save, 159
- data::Serializer< bool >, 160
 - Load, 160
 - Save, 160
- data::Serializer< int >, 160
 - Load, 161
 - Save, 161
- data::Serializer< multimap< string, int > >, 161
 - Load, 162
 - Save, 162
- data::Serializer< string >, 162
 - Load, 162
 - Save, 162
- data::Serializer< T >, 159
- data::Serializer< uint64_t >, 163
 - Load, 163
 - Save, 163
- data::Serializer< vector< T > >, 163
 - Load, 164
 - Save, 164
- data::UnlockedAccess, 179
 - configure, 179
 - fgetc, 179
 - fputc, 179
 - fread, 179
 - fwrite, 179
- data::vint_vector, 180
 - ~vint_vector, 181
 - back, 181
 - clear, 181
 - data, 182
 - data_bytes, 181
 - mask, 182
 - num_bytes, 181
 - num_bytes_, 182
 - operator=, 181
 - operator[], 181
 - push_back, 182
 - set_num_bytes, 182
 - size, 182
 - vint_vector, 181
- data_bytes
 - data::vint_vector, 181
- data_length
 - jpeg2000::Placeholder, 134
- data_ptr
 - AppInfo, 25
- data_writer
 - jipip::DataBinServer, 68
- DataBinClass
 - jipip::DataBinClass, 61
- DataBinServer
 - jipip::DataBinServer, 65
- DataBinWriter
 - jipip::DataBinWriter, 71
- databin_class
 - jipip::DataBinWriter, 76
- Dispose
 - ipc::Event, 79
 - ipc::IPCObject, 118
 - ipc::Mutex, 122
 - ipc::RdWrLock, 147
- EOC_MARKER
 - file_manager.cc, 205
- EOR
 - jipip::EOR, 78
- ERROR
 - trace.h, 219
- EXTENDED_PRECINCT
 - jipip::DataBinClass, 61
- EXTENDED_TILE
 - jipip::DataBinClass, 61
- end
 - jipip::DataBinWriter, 76
- end_woi
 - jipip::DataBinServer, 65
- end_woi_
 - jipip::DataBinServer, 68
- eof
 - jipip::DataBinServer, 68
 - jipip::DataBinWriter, 76
- errorStream
 - TraceSystem, 176
- esa_jipip_server.cc, 195
 - app_info, 196
 - base_id, 196
 - CONFIG_FILE, 196
 - cfg, 196
 - child_address, 196
 - child_lost, 197
 - child_socket, 197
 - ChildProcess, 196
 - ClientThread, 196
 - father_address, 197
 - index_manager, 197
 - main, 196
 - POLLRDHUP, 196
 - ParseArguments, 196
 - poll_table, 197
 - SERVER_APP_NAME, 196
 - SERVER_NAME, 196

- SIGCHLD_handler, 196
- ExistCacheImage
 - jpeg2000::FileManager, 83
- Exists
 - data::BaseFile, 31
- FLST_BOX_ID
 - file_manager.cc, 205
- FTBL_BOX_ID
 - file_manager.cc, 205
- FastFile
 - data, 12
- father_address
 - esa_jpip_server.cc, 197
- father_memory
 - AppInfo, 24
- father_memory_
 - AppInfo, 26
- father_pid
 - AppInfo::Data, 60
- father_sock
 - ClientInfo, 44
- father_sock_
 - ClientInfo, 44
- fds
 - net::PollTable, 142
- fgetc
 - data::LockedAccess, 119
 - data::UnlockedAccess, 179
- File
 - data, 12
- file
 - jpeg2000::IndexManager, 110
- file_
 - data::BaseStream, 39
- file_appender
 - TraceSystem, 176
- file_layout
 - TraceSystem, 176
- file_manager
 - jpeg2000::IndexManager, 110
- file_manager.cc
 - ASOC_BOX_ID, 205
 - COD_MARKER, 205
 - DBTL_BOX_ID, 205
 - EOC_MARKER, 205
 - FLST_BOX_ID, 205
 - FTBL_BOX_ID, 205
 - JP2C_BOX_ID, 205
 - JPCH_BOX_ID, 205
 - NLST_BOX_ID, 205
 - PLT_MARKER, 205
 - SIZ_MARKER, 205
 - SOC_MARKER, 205
 - SOD_MARKER, 205
 - SOT_MARKER, 205
 - URL_BOX_ID, 205
 - XML_BOX_ID, 205
- file_manager_
 - jpeg2000::IndexManager, 112
- file_ptr
 - data::BaseFile, 36
- FileAccess
 - data::InputOperator, 115
 - data::OutputOperator, 124
- FileManager
 - jpeg2000::FileManager, 83
- FileSegment
 - data::FileSegment, 91
- files
 - jpeg2000::IndexManager, 112
- FillTotalPrecinctsVector
 - jpeg2000::CodingParameters, 55
- first
 - jpeg2000::Range, 146
- fputc
 - data::LockedAccess, 119
 - data::UnlockedAccess, 179
- fread
 - data::LockedAccess, 119
 - data::UnlockedAccess, 179
- fsiz
 - jpeg2000::Request::ParametersMask, 131
- full_meta
 - jpeg2000::CacheModel, 42
- fwrite
 - data::LockedAccess, 119
 - data::UnlockedAccess, 179
- GET
 - http::Request, 154
- GenerateChunk
 - jpeg2000::DataBinServer, 65
- Get
 - ipc::Event, 79
 - jpeg2000::DataBinSelector< DataBinClass::MAIN_HE<⇐
ADER >, 62
 - jpeg2000::DataBinSelector< DataBinClass::META_D<⇐
ATA >, 62
 - jpeg2000::DataBinSelector< DataBinClass::PRECIN<⇐
CT >, 63
 - jpeg2000::DataBinSelector< DataBinClass::TILE_HE<⇐
ADER >, 63
- GetBegin
 - jpeg2000::IndexManager, 110
- GetCacheFileName
 - jpeg2000::FileManager, 83
- GetClosestResolution
 - jpeg2000::CodingParameters, 55
- GetCodedChar
 - jpeg2000::Request, 150
- GetCodestream
 - jpeg2000::CacheModel, 41
- GetCodingParameters
 - jpeg2000::ImageIndex, 101
- GetCount
 - jpeg2000::DataBinWriter, 71
- GetCurrentPacket

- jipip::WOIComposer, 186
- GetDataBin
 - jipip::CacheModel, 41
- GetDescriptor
 - data::BaseFile, 31
- GetEnd
 - jpeg2000::IndexManager, 110
- GetFree
 - jipip::DataBinWriter, 71
- GetHyperLink
 - jpeg2000::ImageIndex, 101
- GetIndex
 - jpeg2000::Range, 145
- GetItem
 - jpeg2000::Range, 146
- GetMainHeader
 - jpeg2000::ImageIndex, 101
 - jipip::CacheModel::Codestream, 49
- GetMetadata
 - jpeg2000::ImageIndex, 101
 - jipip::CacheModel, 42
- GetName
 - jipip::DataBinClass, 61
- GetNextPacket
 - jipip::WOIComposer, 186
- GetNumCodestreams
 - jpeg2000::ImageIndex, 103
- GetNumHyperLinks
 - jpeg2000::ImageIndex, 103
- GetNumMetadatas
 - jpeg2000::ImageIndex, 103
- GetOffset
 - data::BaseFile, 31
- GetOffsetPacket
 - jpeg2000::ImageIndex, 103
- GetPLTLength
 - jpeg2000::ImageIndex, 104
- GetPacket
 - jpeg2000::ImageIndex, 103
- GetPath
 - net::InetAddress, 114
 - net::UnixAddress, 178
- GetPathName
 - jpeg2000::ImageIndex, 103
- GetPlaceholder
 - jpeg2000::ImageIndex, 104
- GetPort
 - net::InetAddress, 114
- GetPrecinct
 - jipip::CacheModel::Codestream, 50
- GetPrecinctDataBinId
 - jpeg2000::CodingParameters, 55
- GetPrecincts
 - jpeg2000::CodingParameters, 56
- GetProcStat
 - AppInfo, 24
- GetProcStat_
 - AppInfo, 24
- GetProgressionIndex
 - jpeg2000::CodingParameters, 56
- GetProgressionIndexLRCP
 - jpeg2000::CodingParameters, 56
- GetProgressionIndexRLCP
 - jpeg2000::CodingParameters, 56
- GetProgressionIndexRPCL
 - jpeg2000::CodingParameters, 56
- GetReadBytes
 - net::SocketBuffer, 173
- GetResolution
 - jipip::Request, 150
- GetRoundDownResolution
 - jpeg2000::CodingParameters, 58
- GetRoundUpResolution
 - jpeg2000::CodingParameters, 58
- GetSize
 - data::BaseFile, 31
 - jpeg2000::IndexManager, 110
 - net::Address, 18
 - net::InetAddress, 114
 - net::PollTable, 141
 - net::UnixAddress, 178
- GetSockAddr
 - net::Address, 18
 - net::InetAddress, 114
 - net::UnixAddress, 178
- GetSocket
 - net::SocketBuffer, 173
- GetTileHeader
 - jipip::CacheModel::Codestream, 50
- has_woi
 - jipip::DataBinServer, 69
- HasWOI
 - jipip::Request::ParametersMask, 131
- Header
 - http::Header, 94
- header
 - jpeg2000::CodestreamIndex, 52
 - jpeg2000::Placeholder, 134
 - jipip::CacheModel::Codestream, 50
- HeaderBase
 - http::HeaderBase, 95
 - http::HeaderBase< HeaderName::UNDEFINED >, 96
- http, 12
 - operator<<, 13
 - operator>>, 13
- http/header.cc, 197
- http/header.h, 197
- http/http.h, 198
- http/protocol.cc, 198
- http/protocol.h, 198
- http/request.cc, 198
- http/request.h, 199
- http/response.cc, 200
- http/response.h, 200
- http::Header, 92

- CacheControl, 93
- ContentLength, 93
- ContentType, 93
- Header, 94
- operator==, 94
- TransferEncoding, 93
- http::HeaderBase
 - HeaderBase, 95
 - name, 95
 - operator<<, 95
 - operator>>, 95
 - value, 95
- http::HeaderBase< HeaderName::UNDEFINED >, 96
 - HeaderBase, 96
 - name, 97
 - operator<<, 97
 - operator>>, 97
 - value, 97
- http::HeaderBase< NAME >, 94
- http::HeaderName, 97
 - CACHE_CONTROL, 98
 - CONTENT_LENGTH, 98
 - CONTENT_TYPE, 98
 - TRANSFER_ENCODING, 98
 - UNDEFINED, 98
- http::Protocol, 142
 - CRLF, 144
 - mayorVersion, 143
 - mayorVersion_, 144
 - minorVersion, 143
 - minorVersion_, 144
 - operator<<, 144
 - operator>>, 144
 - Protocol, 143
- http::Request, 153
 - GET, 154
 - object, 157
 - operator<<, 157
 - operator>>, 157
 - POST, 154
 - parameters, 157
 - Parse, 156
 - ParseParameter, 156
 - ParseParameters, 156
 - ParseURI, 156
 - protocol, 157
 - Request, 155
 - Type, 154
 - type, 157
 - UNKNOWN, 154
- http::Response, 157
 - code, 158
 - operator<<, 158
 - operator>>, 158
 - protocol, 158
 - Response, 158
 - StatusCodes, 159
 - statusCodesInitializer, 159
- http::Response::StatusCodesInitializer, 175
 - StatusCodesInitializer, 175
- hyper_links
 - jpeg2000::ImageIndex, 105
- IMAGE_DONE
 - jipip::EOR, 77
- INPUT_BUFFER_LENGTH
 - net::SocketBuffer, 173
- IPCObject
 - ipc::IPCObject, 117
- id
 - jpeg2000::Placeholder, 134
- im_index
 - jipip::DataBinServer, 69
- ImageIndex
 - jpeg2000::ImageIndex, 101
- ImageInfo
 - jpeg2000::ImageInfo, 107
- images_folder
 - AppConfig, 20
- images_folder_
 - AppConfig, 21
- in_buf
 - net::SocketBuffer, 173
- in_len
 - net::SocketBuffer, 173
- index_list
 - jpeg2000::IndexManager, 112
- index_manager
 - ClientManager, 47
 - esa_jpip_server.cc, 197
- IndexManager
 - jpeg2000::ImageIndex, 105
 - jpeg2000::IndexManager, 109
- InetAddress
 - net::InetAddress, 113, 114
- ini
 - jipip::DataBinWriter, 76
- Init
 - AppInfo, 24
 - ipc::Event, 79, 80
 - ipc::IPCObject, 118
 - ipc::Mutex, 122
 - ipc::RdWrLock, 148
 - jpeg2000::FileManager, 83
 - jpeg2000::ImageIndex, 104
 - jpeg2000::IndexManager, 110
- ipc, 13
 - WAIT_ERROR, 14
 - WAIT_OBJECT, 14
 - WAIT_TIMEOUT, 14
 - WaitResult, 14
- ipc/event.cc, 200
- ipc/event.h, 201
- ipc/ipc.h, 201
- ipc/ipc_object.cc, 201
- ipc/ipc_object.h, 201
- ipc/mutex.cc, 202

- ipc/mutex.h, [202](#)
- ipc/rdwr_lock.cc, [202](#)
- ipc/rdwr_lock.h, [203](#)
- ipc::Event, [78](#)
 - condv, [81](#)
 - Dispose, [79](#)
 - Get, [79](#)
 - Init, [79](#), [80](#)
 - manual_reset, [81](#)
 - mutex, [81](#)
 - Ptr, [79](#)
 - Pulse, [80](#)
 - Reset, [80](#)
 - Set, [80](#)
 - state, [81](#)
 - Wait, [81](#)
- ipc::IPCObject, [116](#)
 - ~IPCObject, [117](#)
 - Dispose, [118](#)
 - IPCObject, [117](#)
 - Init, [118](#)
 - IsValid, [118](#)
 - Ptr, [117](#)
 - valid, [118](#)
 - Wait, [118](#)
- ipc::Mutex, [121](#)
 - Dispose, [122](#)
 - Init, [122](#)
 - locker, [123](#)
 - mutex, [123](#)
 - Ptr, [122](#)
 - Release, [122](#)
 - Wait, [123](#)
- ipc::RdWrLock, [147](#)
 - Dispose, [147](#)
 - Init, [148](#)
 - Ptr, [147](#)
 - Release, [148](#)
 - rwlock, [148](#)
 - Wait, [148](#)
 - WaitForWriting, [148](#)
- is_jp2c
 - jpeg2000::Placeholder, [134](#)
- is_running
 - AppInfo, [25](#)
- is_running_
 - AppInfo, [26](#)
- IsBlockingMode
 - net::Socket, [167](#)
- IsContiguousTo
 - data::FileSegment, [91](#)
- IsEOF
 - data::BaseFile, [31](#)
- IsFullMetadata
 - jpip::CacheModel, [42](#)
- IsHyperLinked
 - jpeg2000::ImageIndex, [105](#)
- IsResolutionProgression
 - jpeg2000::CodingParameters, [58](#)
- IsValid
 - data::BaseFile, [31](#)
 - ipc::IPCObject, [118](#)
 - jpeg2000::Range, [146](#)
 - net::Socket, [167](#)
- items
 - jpip::Request::ParametersMask, [131](#)
- JP2C_BOX_ID
 - file_manager.cc, [205](#)
- JPCH_BOX_ID
 - file_manager.cc, [205](#)
- jpeg2000, [14](#)
 - Size, [15](#)
- jpeg2000/codestream_index.cc, [203](#)
- jpeg2000/codestream_index.h, [203](#)
- jpeg2000/coding_parameters.cc, [203](#)
- jpeg2000/coding_parameters.h, [204](#)
- jpeg2000/file_manager.cc, [204](#)
- jpeg2000/file_manager.h, [205](#)
- jpeg2000/image_index.cc, [206](#)
- jpeg2000/image_index.h, [206](#)
- jpeg2000/image_info.cc, [206](#)
- jpeg2000/image_info.h, [207](#)
- jpeg2000/index_manager.cc, [207](#)
- jpeg2000/index_manager.h, [207](#)
- jpeg2000/jpeg2000.h, [208](#)
- jpeg2000/meta_data.cc, [208](#)
- jpeg2000/meta_data.h, [208](#)
- jpeg2000/packet.cc, [208](#)
- jpeg2000/packet.h, [208](#)
- jpeg2000/packet_index.cc, [209](#)
- jpeg2000/packet_index.h, [209](#)
- jpeg2000/place_holder.cc, [209](#)
- jpeg2000/place_holder.h, [209](#)
- jpeg2000/point.cc, [210](#)
- jpeg2000/point.h, [210](#)
- jpeg2000/range.cc, [210](#)
- jpeg2000/range.h, [210](#)
- jpeg2000::CodestreamIndex, [51](#)
 - ~CodestreamIndex, [52](#)
 - Clear, [52](#)
 - CodestreamIndex, [52](#)
 - header, [52](#)
 - operator<<, [52](#)
 - operator=, [52](#)
 - PLT_markers, [52](#)
 - packets, [52](#)
 - SerializeWith, [52](#)
- jpeg2000::CodingParameters, [53](#)
 - ~CodingParameters, [55](#)
 - CPRL_PROGRESSION, [55](#)
 - CodingParameters, [55](#)
 - FillTotalPrecinctsVector, [55](#)
 - GetClosestResolution, [55](#)
 - GetPrecinctDataBinId, [55](#)
 - GetPrecincts, [56](#)
 - GetProgressionIndex, [56](#)

- GetProgressionIndexLRCP, 56
- GetProgressionIndexRLCP, 56
- GetProgressionIndexRPCL, 56
- GetRoundDownResolution, 58
- GetRoundUpResolution, 58
- IsResolutionProgression, 58
- LRCP_PROGRESSION, 55
- num_components, 58
- num_layers, 58
- num_levels, 59
- operator<<, 58
- operator=, 58
- PCRL_PROGRESSION, 55
- precinct_size, 59
- progression, 59
- Ptr, 54
- RLCP_PROGRESSION, 55
- RPCL_PROGRESSION, 55
- SerializeWith, 58
- size, 59
- total_precincts, 59
- jpeg2000::FileManager, 81
 - ~FileManager, 83
 - cache_dir, 83
 - cache_dir_, 89
 - ExistCacheImage, 83
 - FileManager, 83
 - GetCacheFileName, 83
 - Init, 83
 - ReadBoxHeader, 85
 - ReadCODMarker, 85
 - ReadCodestream, 85
 - ReadFlstBox, 86
 - ReadImage, 86
 - ReadJP2, 86
 - ReadJPX, 86
 - ReadNlstBox, 87
 - ReadPLTMarker, 87
 - ReadSIZMarker, 87
 - ReadSODMarker, 87
 - ReadSOTMarker, 89
 - ReadUrlBox, 89
 - root_dir, 89
 - root_dir_, 89
- jpeg2000::ImageIndex, 98
 - ~ImageIndex, 101
 - BuildIndex, 101
 - codestreams, 105
 - coding_parameters, 105
 - GetCodingParameters, 101
 - GetHyperLink, 101
 - GetMainHeader, 101
 - GetMetadata, 101
 - GetNumCodestreams, 103
 - GetNumHyperLinks, 103
 - GetNumMetadatas, 103
 - GetOffsetPacket, 103
 - GetPLTLength, 104
 - GetPacket, 103
 - GetPathName, 103
 - GetPlaceholder, 104
 - hyper_links, 105
 - ImageIndex, 101
 - IndexManager, 105
 - Init, 104
 - IsHyperLinked, 105
 - last_offset_PLT, 106
 - last_offset_packet, 106
 - last_packet, 106
 - last_plt, 106
 - max_resolution, 106
 - meta_data, 106
 - num_references, 106
 - operator CodingParameters::Ptr, 105
 - operator<<, 105
 - operator=, 105
 - packet_indexes, 106
 - path_name, 106
 - Ptr, 100
 - rdwr_lock, 106
 - ReadLock, 105
 - ReadUnlock, 105
- jpeg2000::ImageInfo, 106
 - ~ImageInfo, 107
 - codestreams, 108
 - coding_parameters, 108
 - ImageInfo, 107
 - meta_data, 108
 - meta_data_hyperlinks, 108
 - operator<<, 108
 - operator=, 108
 - paths, 108
 - SerializeWith, 108
- jpeg2000::IndexManager, 108
 - ~IndexManager, 109
 - ClosedImage, 110
 - file_manager, 110
 - file_manager_, 112
 - GetBegin, 110
 - GetEnd, 110
 - GetSize, 110
 - index_list, 112
 - IndexManager, 109
 - Init, 110
 - mutex, 112
 - OpenImage, 110
 - UnsafeClosedImage, 112
 - UnsafeOpenImage, 112
- jpeg2000::Metadata, 119
 - ~Metadata, 120
 - meta_data, 121
 - Metadata, 120
 - operator<<, 120
 - operator=, 120
 - place_holders, 121
 - SerializeWith, 120

- jpeg2000::Packet, 125
 - ~Packet, 126
 - component, 126
 - layer, 126
 - operator<<, 126
 - operator=, 126
 - Packet, 126
 - precinct_xy, 126
 - resolution, 126
- jpeg2000::PacketIndex, 127
 - ~PacketIndex, 128
 - Add, 128
 - aux, 130
 - Clear, 128
 - MINIMUM_OFFSET, 128
 - offsets, 130
 - operator=, 128
 - operator[], 128
 - PacketIndex, 128
 - Size, 130
- jpeg2000::Placeholder, 132
 - ~Placeholder, 133
 - data_length, 134
 - header, 134
 - id, 134
 - is_jp2c, 134
 - length, 133
 - operator<<, 134
 - operator=, 133
 - Placeholder, 133
 - SerializeWith, 133
- jpeg2000::Point, 134
 - ~Point, 137
 - operator!=, 138
 - operator<<, 139
 - operator*, 138
 - operator*=, 137
 - operator+, 138
 - operator++, 137
 - operator+=, 137
 - operator-, 138, 139
 - operator--, 137
 - operator-=, 137
 - operator/, 139
 - operator/=: 138
 - operator=, 138
 - operator==, 139
 - Point, 135, 137
 - SerializeWith, 138
 - x, 139
 - y, 139
- jpeg2000::Range, 144
 - ~Range, 145
 - first, 146
 - GetIndex, 145
 - GetItem, 146
 - IsValid, 146
 - last, 146
 - Length, 146
 - operator!=, 146
 - operator<<, 146
 - operator=, 146
 - operator==, 146
 - Range, 145
- jpeg, 15
- jpeg/cache_model.cc, 211
- jpeg/cache_model.h, 211
- jpeg/databin_server.cc, 211
- jpeg/databin_server.h, 212
- jpeg/databin_writer.cc, 212
- jpeg/databin_writer.h, 212
- jpeg/jpip.cc, 213
- jpeg/jpip.h, 213
- jpeg/request.cc, 199
- jpeg/request.h, 199
- jpeg/woi.cc, 213
- jpeg/woi.h, 213
- jpeg/woi_composer.cc, 214
- jpeg/woi_composer.h, 214
- jpeg::CacheModel, 39
 - ~CacheModel, 40
 - AddToDataBin, 41
 - AddToMetadata, 41
 - CacheModel, 40
 - Clear, 41
 - codestreams, 42
 - full_meta, 42
 - GetCodestream, 41
 - GetDataBin, 41
 - GetMetadata, 42
 - IsFullMetadata, 42
 - meta_data, 43
 - operator+=, 42
 - operator=, 42
 - Pack, 42
 - SerializeWith, 42
 - SetFullMetadata, 42
- jpeg::CacheModel::Codestream, 48
 - AddToMainHeader, 49
 - AddToPrecinct, 49
 - AddToTileHeader, 49
 - Codestream, 49
 - GetMainHeader, 49
 - GetPrecinct, 50
 - GetTileHeader, 50
 - header, 50
 - min_precinct, 50
 - operator+=, 50
 - operator=, 50
 - Pack, 50
 - precincts, 50
 - SerializeWith, 50
 - tile_header, 51
- jpeg::DataBinClass, 60
 - DataBinClass, 61
 - EXTENDED_PRECINCT, 61

- EXTENDED_TILE, 61
- GetName, 61
- MAIN_HEADER, 61
- META_DATA, 61
- PRECINCT, 61
- TILE_DATA, 61
- TILE_HEADER, 61
- jipip::DataBinSelector< BIN_CLASS >, 61
- jipip::DataBinSelector< DataBinClass::MAIN_HEADER >, 62
 - AddTo, 62
 - Get, 62
- jipip::DataBinSelector< DataBinClass::META_DATA >, 62
 - AddTo, 62
 - Get, 62
- jipip::DataBinSelector< DataBinClass::PRECINCT >, 63
 - AddTo, 63
 - Get, 63
- jipip::DataBinSelector< DataBinClass::TILE_HEADER >, 63
 - AddTo, 63
 - Get, 63
- jipip::DataBinServer, 64
 - ~DataBinServer, 65
 - cache_model, 68
 - current_idx, 68
 - data_writer, 68
 - DataBinServer, 65
 - end_woi, 65
 - end_woi_, 68
 - eof, 68
 - file, 69
 - files, 69
 - GenerateChunk, 65
 - has_woi, 69
 - im_index, 69
 - MINIMUM_SPACE, 65
 - metareq, 69
 - pending, 69
 - range, 69
 - Reset, 67
 - SetRequest, 67
 - woi, 69
 - woi_composer, 69
 - WritePlaceholder, 67
 - WriteSegment, 68
- jipip::DataBinWriter, 69
 - ~DataBinWriter, 71
 - ClearPreviousIds, 71
 - codestream_idx, 76
 - DataBinWriter, 71
 - databin_class, 76
 - end, 76
 - eof, 76
 - GetCount, 71
 - GetFree, 71
 - ini, 76
 - operator bool, 71
 - prev_codestream_idx, 77
 - prev_databin_class, 77
 - ptr, 77
 - SetBuffer, 71
 - SetCodestream, 73
 - SetDataBinClass, 73
 - Write, 73
 - WriteEOR, 75
 - WriteEmpty, 73
 - WriteHeader, 75
 - WritePlaceholder, 75
 - WriteVBAS, 76
 - WriteValue, 76
- jipip::EOR, 77
 - BYTE_LIMIT_REACHED, 78
 - EOR, 78
 - IMAGE_DONE, 77
 - NON_SPECIFIED, 78
 - QUALITY_LIMIT_REACHED, 78
 - RESPONSE_LIMIT_REACHED, 78
 - SESSION_LIMIT_REACHED, 78
 - WINDOW_CHANGE, 78
 - WINDOW_DONE, 77
- jipip::Request, 149
 - ~Request, 150
 - CLOSEST, 150
 - cache_model, 152
 - GetCodedChar, 150
 - GetResolution, 150
 - length_response, 152
 - mask, 152
 - max_codestream, 153
 - min_codestream, 153
 - ParseModel, 152
 - ParseParameter, 152
 - ParseParameters, 152
 - ROUNDDOWN, 150
 - ROUNDUP, 150
 - Request, 150
 - resolution_size, 153
 - round_direction, 153
 - RoundDirection, 150
 - woi_position, 153
 - woi_size, 153
- jipip::Request::ParametersMask, 130
 - cclose, 131
 - cid, 131
 - Clear, 131
 - cnew, 131
 - context, 131
 - fsiz, 131
 - HasWOI, 131
 - items, 131
 - len, 131
 - metareq, 132
 - model, 132

- ParametersMask, 131
- roff, 132
- rsiz, 132
- stream, 132
- target, 132
- value, 132
- jpip::WOI, 182
 - ~WOI, 184
 - operator!=, 184
 - operator<<, 184
 - operator=, 184
 - operator==, 184
 - position, 184
 - resolution, 184
 - size, 184
 - WOI, 183, 184
- jpip::WOIComposer, 185
 - ~WOIComposer, 186
 - coding_parameters, 187
 - current_packet, 187
 - GetCurrentPacket, 186
 - GetNextPacket, 186
 - max_precinct_xy, 187
 - max_resolution, 187
 - min_precinct_xy, 187
 - more_packets, 187
 - operator=, 186
 - pxy1, 187
 - pxy2, 187
 - Reset, 186
 - WOIComposer, 186
- LOCK_FILE
 - app_info.cc, 190
- LOG
 - trace.h, 219
- LOG4CPP_FIX_ERROR_COLLISION
 - trace.h, 219
- LOGC
 - trace.h, 219
- LRCP_PROGRESSION
 - jpeg2000::CodingParameters, 55
- last
 - jpeg2000::Range, 146
- last_offset_PLT
 - jpeg2000::ImageIndex, 106
- last_offset_packet
 - jpeg2000::ImageIndex, 106
- last_packet
 - jpeg2000::ImageIndex, 106
- last_plt
 - jpeg2000::ImageIndex, 106
- layer
 - jpeg2000::Packet, 126
- layout
 - TraceSystem, 176
- len
 - jpip::Request::ParametersMask, 131
- Length
 - jpeg2000::Range, 146
- length
 - data::FileSegment, 92
 - jpeg2000::Placeholder, 133
- length_response
 - jpip::Request, 152
- ListenAt
 - net::Socket, 167
- Load
 - AppConfig, 20
 - data::Serializer, 159
 - data::Serializer< bool >, 160
 - data::Serializer< int >, 161
 - data::Serializer< multimap< string, int > >, 162
 - data::Serializer< string >, 162
 - data::Serializer< uint64_t >, 163
 - data::Serializer< vector< T > >, 164
- lock_file
 - AppInfo, 26
- locker
 - ipc::Mutex, 123
- log_requests
 - AppConfig, 20
- log_requests_
 - AppConfig, 22
- logStream
 - TraceSystem, 176
- logging
 - AppConfig, 20
- logging_
 - AppConfig, 22
- logging_folder
 - AppConfig, 21
- logging_folder_
 - AppConfig, 22
- MAIN_HEADER
 - jpip::DataBinClass, 61
- META_DATA
 - jpip::DataBinClass, 61
- MINIMUM_OFFSET
 - jpeg2000::PacketIndex, 128
- MINIMUM_SPACE
 - jpip::DataBinServer, 65
- main
 - esa_jpip_server.cc, 196
 - packet_information.cc, 217
- manual_reset
 - ipc::Event, 81
- mask
 - data::vint_vector, 182
 - jpip::Request, 152
- max_chunk_size
 - AppConfig, 21
- max_chunk_size_
 - AppConfig, 22
- max_codestream
 - jpip::Request, 153
- max_connections

- AppConfig, 21
- max_connections_
 - AppConfig, 22
- max_precinct_xy
 - jpeg2000::WOIComposer, 187
- max_resolution
 - jpeg2000::ImageIndex, 106
 - jpeg2000::WOIComposer, 187
- mayorVersion
 - http::Protocol, 143
- mayorVersion_
 - http::Protocol, 144
- meta_data
 - jpeg2000::ImageIndex, 106
 - jpeg2000::ImageInfo, 108
 - jpeg2000::Metadata, 121
 - jpeg2000::CacheModel, 43
- meta_data_hyperlinks
 - jpeg2000::ImageInfo, 108
- Metadata
 - jpeg2000::Metadata, 120
- metareq
 - jpeg2000::DataBinServer, 69
 - jpeg2000::Request::ParametersMask, 132
- min_codestream
 - jpeg2000::Request, 153
- min_precinct
 - jpeg2000::CacheModel::Codestream, 50
- min_precinct_xy
 - jpeg2000::WOIComposer, 187
- minorVersion
 - http::Protocol, 143
- minorVersion_
 - http::Protocol, 144
- model
 - jpeg2000::Request::ParametersMask, 132
- more_packets
 - jpeg2000::WOIComposer, 187
- mutex
 - ipc::Event, 81
 - ipc::Mutex, 123
 - jpeg2000::IndexManager, 112
- NLST_BOX_ID
 - file_manager.cc, 205
- NON_SPECIFIED
 - jpeg2000::EOR, 78
- name
 - http::HeaderBase, 95
 - http::HeaderBase< HeaderName::UNDEFINED >, 97
- net, 16
- net/address.cc, 214
- net/address.h, 214
- net/net.h, 215
- net/poll_table.cc, 215
- net/poll_table.h, 215
- net/socket.cc, 215
- net/socket.h, 216
- net/socket_stream.cc, 217
- net/socket_stream.h, 217
- net::Address, 17
 - ~Address, 17
 - Address, 17
 - GetSize, 18
 - GetSockAddr, 18
- net::InetAddress, 113
 - GetPath, 114
 - GetPort, 114
 - GetSize, 114
 - GetSockAddr, 114
 - InetAddress, 113, 114
 - operator=, 114
 - sock_addr, 115
- net::PollFD, 139
 - operator==, 140
 - PollFD, 140
- net::PollTable, 140
 - ~PollTable, 141
 - Add, 141
 - fds, 142
 - GetSize, 141
 - operator[], 141
 - Poll, 141
 - PollTable, 141
 - Remove, 142
 - RemoveAt, 142
- net::Socket, 164
 - ~Socket, 166
 - Accept, 166
 - BindTo, 166
 - Close, 166
 - ConnectTo, 166
 - IsBlockingMode, 167
 - IsValid, 167
 - ListenAt, 167
 - OpenInet, 167
 - OpenUnix, 167
 - operator int, 169
 - operator=, 169
 - Receive, 169
 - ReceiveDescriptor, 169
 - ReceiveFrom, 169
 - Send, 170
 - SendDescriptor, 170
 - SendTo, 170
 - SetBlockingMode, 171
 - SetNoDelay, 171
 - sid, 172
 - Socket, 165, 166
 - WaitForInput, 171
 - WaitForOutput, 171
- net::SocketBuffer, 172
 - ~SocketBuffer, 173
 - GetReadBytes, 173
 - GetSocket, 173
 - INPUT_BUFFER_LENGTH, 173

- in_buf, 173
- in_len, 173
- OUTPUT_BUFFER_LENGTH, 173
- out_buf, 173
- out_len, 173
- overflow, 173
- socket, 174
- SocketBuffer, 173
- sum, 174
- sync, 173
- underflow, 173
- net::SocketStream, 174
 - ~SocketStream, 174
 - operator->, 174
 - SocketStream, 174
- net::UnixAddress, 177
 - GetPath, 178
 - GetSize, 178
 - GetSockAddr, 178
 - operator=, 178
 - Reset, 178
 - sock_addr, 179
 - UnixAddress, 178
- Null
 - data::FileSegment, 92
- num_bytes
 - data::vint_vector, 181
- num_bytes_
 - data::vint_vector, 182
- num_components
 - jpeg2000::CodingParameters, 58
- num_connections
 - AppInfo::Data, 60
- num_layers
 - jpeg2000::CodingParameters, 58
- num_levels
 - jpeg2000::CodingParameters, 59
- num_references
 - jpeg2000::ImageIndex, 106
- num_threads
 - AppInfo, 25
- num_threads_
 - AppInfo, 26
- OUTPUT_BUFFER_LENGTH
 - net::SocketBuffer, 173
- object
 - http::Request, 157
- offset
 - data::FileSegment, 92
- offsets
 - jpeg2000::PacketIndex, 130
- Open
 - data::BaseFile, 31, 33
 - data::BaseStream, 38
- OpenForReading
 - data::BaseFile, 33
- OpenForWriting
 - data::BaseFile, 33
- OpenImage
 - jpeg2000::IndexManager, 110
- OpenInet
 - net::Socket, 167
- OpenUnix
 - net::Socket, 167
- operator bool
 - data::BaseFile, 33
 - data::BaseStream, 38
 - jpeg2000::DataBinWriter, 71
- operator CodingParameters::Ptr
 - jpeg2000::ImageIndex, 105
- operator int
 - net::Socket, 169
- operator!=
 - data::FileSegment, 91
 - jpeg2000::Point, 138
 - jpeg2000::Range, 146
 - jpeg2000::WOL, 184
- operator<<
 - AppConfig, 21
 - AppInfo, 25
 - data::FileSegment, 92
 - http, 13
 - http::HeaderBase, 95
 - http::HeaderBase< HeaderName::UNDEFINED
>, 97
 - http::Protocol, 144
 - http::Request, 157
 - http::Response, 158
 - jpeg2000::CodestreamIndex, 52
 - jpeg2000::CodingParameters, 58
 - jpeg2000::ImageIndex, 105
 - jpeg2000::ImageInfo, 108
 - jpeg2000::Metadata, 120
 - jpeg2000::Packet, 126
 - jpeg2000::Placeholder, 134
 - jpeg2000::Point, 139
 - jpeg2000::Range, 146
 - jpeg2000::WOL, 184
- operator>>
 - http, 13
 - http::HeaderBase, 95
 - http::HeaderBase< HeaderName::UNDEFINED
>, 97
 - http::Protocol, 144
 - http::Request, 157
 - http::Response, 158
- operator*
 - jpeg2000::Point, 138
- operator*=
 - jpeg2000::Point, 137
- operator+
 - jpeg2000::Point, 138
- operator++
 - jpeg2000::Point, 137
- operator+=
 - jpeg2000::Point, 137

- jpip::CacheModel, 42
 - jpip::CacheModel::Codestream, 50
- operator-
 - jpeg2000::Point, 138, 139
- operator->
 - AppInfo, 25
 - net::SocketStream, 174
- operator--
 - jpeg2000::Point, 137
- operator=
 - jpeg2000::Point, 137
- operator/
 - jpeg2000::Point, 139
- operator/=
 - jpeg2000::Point, 138
- operator=
 - data::FileSegment, 91
 - data::vint_vector, 181
 - jpeg2000::CodestreamIndex, 52
 - jpeg2000::CodingParameters, 58
 - jpeg2000::ImageIndex, 105
 - jpeg2000::ImageInfo, 108
 - jpeg2000::Metadata, 120
 - jpeg2000::Packet, 126
 - jpeg2000::PacketIndex, 128
 - jpeg2000::Placeholder, 133
 - jpeg2000::Point, 138
 - jpeg2000::Range, 146
 - jpip::CacheModel, 42
 - jpip::CacheModel::Codestream, 50
 - jpip::WOI, 184
 - jpip::WOIComposer, 186
 - net::InetAddress, 114
 - net::Socket, 169
 - net::UnixAddress, 178
- operator==
 - data::FileSegment, 91
 - http::Header, 94
 - jpeg2000::Point, 139
 - jpeg2000::Range, 146
 - jpip::WOI, 184
 - net::PollFD, 140
- operator&
 - data::BaseStream, 38
- operator[]
 - data::vint_vector, 181
 - jpeg2000::PacketIndex, 128
 - net::PollTable, 141
- out_buf
 - net::SocketBuffer, 173
- out_len
 - net::SocketBuffer, 173
- overflow
 - net::SocketBuffer, 173
- PCRL_PROGRESSION
 - jpeg2000::CodingParameters, 55
- PLT_MARKER
 - file_manager.cc, 205
- PLT_markers
 - jpeg2000::CodestreamIndex, 52
- POLLRDHUP
 - esa_jpip_server.cc, 196
 - socket.cc, 216
- POST
 - http::Request, 154
- PRECINCT
 - jpip::DataBinClass, 61
- Pack
 - jpip::CacheModel, 42
 - jpip::CacheModel::Codestream, 50
- Packet
 - jpeg2000::Packet, 126
- packet_indexes
 - jpeg2000::ImageIndex, 106
- packet_information.cc, 217
 - main, 217
- PacketIndex
 - jpeg2000::PacketIndex, 128
- packets
 - jpeg2000::CodestreamIndex, 52
- parameters
 - http::Request, 157
- ParametersMask
 - jpip::Request::ParametersMask, 131
- Parse
 - ArgsParser, 27
 - http::Request, 156
- ParseArguments
 - esa_jpip_server.cc, 196
- ParseModel
 - jpip::Request, 152
- ParseParameter
 - http::Request, 156
 - jpip::Request, 152
- ParseParameters
 - http::Request, 156
 - jpip::Request, 152
- ParseURI
 - http::Request, 156
- path_name
 - jpeg2000::ImageIndex, 106
- paths
 - jpeg2000::ImageInfo, 108
- pending
 - jpip::DataBinServer, 69
- place_holders
 - jpeg2000::Metadata, 121
- Placeholder
 - jpeg2000::Placeholder, 133
- Point
 - jpeg2000::Point, 135, 137
- Poll
 - net::PollTable, 141
- poll_table
 - esa_jpip_server.cc, 197
- PollFD

- net::PollFD, 140
- PollTable
 - net::PollTable, 141
- port
 - AppConfig, 21
- port_
 - AppConfig, 22
- position
 - jpeg2000::CodingParameters, 59
- precinct_size
 - jpeg2000::CodingParameters, 59
- precinct_xy
 - jpeg2000::Packet, 126
- precincts
 - jpeg2000::CacheModel::Codestream, 50
- prev_codestream_idx
 - jpeg2000::DataBinWriter, 77
- prev_databin_class
 - jpeg2000::DataBinWriter, 77
- progression
 - jpeg2000::CodingParameters, 59
- Protocol
 - http::Protocol, 143
- protocol
 - http::Request, 157
 - http::Response, 158
- Ptr
 - data::BaseFile, 30
 - ipc::Event, 79
 - ipc::IPCObject, 117
 - ipc::Mutex, 122
 - ipc::RdWrLock, 147
 - jpeg2000::CodingParameters, 54
 - jpeg2000::ImageIndex, 100
- ptr
 - jpeg2000::DataBinWriter, 77
- Pulse
 - ipc::Event, 80
- push_back
 - data::vint_vector, 182
- pxy1
 - jpeg2000::WOIComposer, 187
- pxy2
 - jpeg2000::WOIComposer, 187
- QUALITY_LIMIT_REACHED
 - jpeg2000::EOR, 78
- RESPONSE_LIMIT_REACHED
 - jpeg2000::EOR, 78
- RLCP_PROGRESSION
 - jpeg2000::CodingParameters, 55
- ROUNDDOWN
 - jpeg2000::Request, 150
- ROUNDUP
 - jpeg2000::Request, 150
- RPCL_PROGRESSION
 - jpeg2000::CodingParameters, 55
- Range
 - jpeg2000::Range, 145
- range
 - jpeg2000::DataBinServer, 69
- rdwr_lock
 - jpeg2000::ImageIndex, 106
- Read
 - data::BaseFile, 34
- read
 - ui, 177
- ReadBoxHeader
 - jpeg2000::FileManager, 85
- ReadByte
 - data::BaseFile, 34
- ReadCODMarker
 - jpeg2000::FileManager, 85
- ReadCodestream
 - jpeg2000::FileManager, 85
- ReadFlstBox
 - jpeg2000::FileManager, 86
- ReadImage
 - jpeg2000::FileManager, 86
- ReadJP2
 - jpeg2000::FileManager, 86
- ReadJPX
 - jpeg2000::FileManager, 86
- ReadLock
 - jpeg2000::ImageIndex, 105
- ReadNlstBox
 - jpeg2000::FileManager, 87
- ReadPLTMarker
 - jpeg2000::FileManager, 87
- ReadReverse
 - data::BaseFile, 34
- ReadSIZMarker
 - jpeg2000::FileManager, 87
- ReadSODMarker
 - jpeg2000::FileManager, 87
- ReadSOTMarker
 - jpeg2000::FileManager, 89
- ReadUnlock
 - jpeg2000::ImageIndex, 105
- ReadUrlBox
 - jpeg2000::FileManager, 89
- Receive
 - net::Socket, 169
- ReceiveDescriptor
 - net::Socket, 169
- ReceiveFrom
 - net::Socket, 169
- Release
 - ipc::Mutex, 122
 - ipc::RdWrLock, 148
- Remove
 - net::PollTable, 142
- RemoveAt
 - net::PollTable, 142
- RemoveFirst
 - data::FileSegment, 91

- RemoveLast
 - data::FileSegment, [92](#)
- Request
 - http::Request, [155](#)
 - jpip::Request, [150](#)
- Reset
 - AppInfo::Data, [60](#)
 - ipc::Event, [80](#)
 - jpip::DataBinServer, [67](#)
 - jpip::WOIComposer, [186](#)
 - net::UnixAddress, [178](#)
- resolution
 - jpeg2000::Packet, [126](#)
 - jpip::WOI, [184](#)
- resolution_size
 - jpip::Request, [153](#)
- Response
 - http::Response, [158](#)
- result
 - data::BaseStream, [38](#)
- result_
 - data::BaseStream, [39](#)
- roff
 - jpip::Request::ParametersMask, [132](#)
- root_dir
 - jpeg2000::FileManager, [89](#)
- root_dir_
 - jpeg2000::FileManager, [89](#)
- round_direction
 - jpip::Request, [153](#)
- RoundDirection
 - jpip::Request, [150](#)
- rsiz
 - jpip::Request::ParametersMask, [132](#)
- Run
 - ClientManager, [47](#)
- RunBasic
 - ClientManager, [47](#)
- rwlock
 - ipc::RdWrLock, [148](#)
- SERVER_APP_NAME
 - esa_jpip_server.cc, [196](#)
- SERVER_NAME
 - esa_jpip_server.cc, [196](#)
- SESSION_LIMIT_REACHED
 - jpip::EOR, [78](#)
- SHARED_PTR
 - tr1_compat.h, [218](#)
- SIGCHLD_handler
 - esa_jpip_server.cc, [196](#)
- SIZ_MARKER
 - file_manager.cc, [205](#)
- SOC_MARKER
 - file_manager.cc, [205](#)
- SOD_MARKER
 - file_manager.cc, [205](#)
- SOT_MARKER
 - file_manager.cc, [205](#)
- Save
 - data::Serializer, [159](#)
 - data::Serializer< bool >, [160](#)
 - data::Serializer< int >, [161](#)
 - data::Serializer< multimap< string, int > >, [162](#)
 - data::Serializer< string >, [162](#)
 - data::Serializer< uint64_t >, [163](#)
 - data::Serializer< vector< T > >, [164](#)
- Seek
 - data::BaseFile, [34](#)
- Send
 - net::Socket, [170](#)
- SendDescriptor
 - net::Socket, [170](#)
- SendTo
 - net::Socket, [170](#)
- Serialize
 - data::InputStream, [116](#)
 - data::OutputStream, [125](#)
- SerializeBytes
 - data::BaseStream, [38](#)
 - data::InputOperator, [115](#)
 - data::OutputOperator, [124](#)
- SerializeWith
 - data::FileSegment, [92](#)
 - jpeg2000::CodestreamIndex, [52](#)
 - jpeg2000::CodingParameters, [58](#)
 - jpeg2000::ImageInfo, [108](#)
 - jpeg2000::Metadata, [120](#)
 - jpeg2000::Placeholder, [133](#)
 - jpeg2000::Point, [138](#)
 - jpip::CacheModel, [42](#)
 - jpip::CacheModel::Codestream, [50](#)
- Set
 - ipc::Event, [80](#)
- set_num_bytes
 - data::vint_vector, [182](#)
- SetBlockingMode
 - net::Socket, [171](#)
- SetBuffer
 - jpip::DataBinWriter, [71](#)
- SetCodestream
 - jpip::DataBinWriter, [73](#)
- SetDataBinClass
 - jpip::DataBinWriter, [73](#)
- SetFullMetadata
 - jpip::CacheModel, [42](#)
- SetNoDelay
 - net::Socket, [171](#)
- SetRequest
 - jpip::DataBinServer, [67](#)
- shmid
 - AppInfo, [26](#)
- sid
 - net::Socket, [172](#)
- Size
 - jpeg2000, [15](#)
 - jpeg2000::PacketIndex, [130](#)

- size
 - data::vint_vector, 182
 - jpeg2000::CodingParameters, 59
 - jpeg::WOI, 184
- sock
 - ClientInfo, 44
- sock_
 - ClientInfo, 45
- sock_addr
 - net::InetAddress, 115
 - net::UnixAddress, 179
- Socket
 - net::Socket, 165, 166
- socket
 - net::SocketBuffer, 174
- socket.cc
 - POLLRDHUP, 216
- SocketBuffer
 - net::SocketBuffer, 173
- SocketStream
 - net::SocketStream, 174
- state
 - ipc::Event, 81
- StatusCodes
 - http::Response, 159
- StatusCodesInitializer
 - http::Response::StatusCodesInitializer, 175
- statusCodesInitializer
 - http::Response, 159
- stream
 - jpeg::Request::ParametersMask, 132
- sum
 - net::SocketBuffer, 174
- sync
 - net::SocketBuffer, 173
- TILE_DATA
 - jpeg::DataBinClass, 61
- TILE_HEADER
 - jpeg::DataBinClass, 61
- TRACE
 - trace.h, 219
- TRANSFER_ENCODING
 - http::HeaderName, 98
- target
 - jpeg::Request::ParametersMask, 132
- tile_header
 - jpeg::CacheModel::Codestream, 51
- time
 - AppInfo, 25
 - ClientInfo, 44
- time_
 - AppInfo, 26
- tm_start
 - ClientInfo, 45
- to_string
 - base, 28
- total_precincts
 - jpeg2000::CodingParameters, 59
- tr1_compat.h, 218
 - SHARED_PTR, 218
- trace.cc, 218
- trace.h, 218
 - _BLUE, 219
 - _GREEN, 219
 - _RED, 219
 - _RESET_COLOR, 219
 - _SET_COLOR, 219
 - _YELLOW, 219
 - CERR, 219
 - ERROR, 219
 - LOG, 219
 - LOG4CPP_FIX_ERROR_COLLISION, 219
 - LOGC, 219
 - TRACE, 219
- traceStream
 - TraceSystem, 176
- TraceSystem, 175
 - ~TraceSystem, 176
 - AppendToFile, 176
 - AppendToFile_, 176
 - appender, 176
 - category, 176
 - errorStream, 176
 - file_appender, 176
 - file_layout, 176
 - layout, 176
 - logStream, 176
 - traceStream, 176
 - TraceSystem, 176
 - traceSystem, 176
- traceSystem
 - TraceSystem, 176
- TransferEncoding
 - http::Header, 93
- Type
 - http::Request, 154
- type
 - http::Request, 157
- UNDEFINED
 - http::HeaderName, 98
- UNKNOWN
 - http::Request, 154
- URL__BOX_ID
 - file_manager.cc, 205
- ui, 177
 - read, 177
- underflow
 - net::SocketBuffer, 173
- UnixAddress
 - net::UnixAddress, 178
- UnsafeClosedImage
 - jpeg2000::IndexManager, 112
- UnsafeOpenImage
 - jpeg2000::IndexManager, 112
- Update
 - AppInfo, 25

VERSION
 version.h, [219](#)
 valid
 ipc::IPCObject, [118](#)
 value
 http::HeaderBase, [95](#)
 http::HeaderBase< HeaderName::UNDEFINED
 >, [97](#)
 jpip::Request::ParametersMask, [132](#)
 version.h, [219](#)
 VERSION, [219](#)
 vint_vector
 data::vint_vector, [181](#)

 WAIT_ERROR
 ipc, [14](#)
 WAIT_OBJECT
 ipc, [14](#)
 WAIT_TIMEOUT
 ipc, [14](#)
 WINDOW_CHANGE
 jpip::EOR, [78](#)
 WINDOW_DONE
 jpip::EOR, [77](#)
 WOI
 jpip::WOI, [183](#), [184](#)
 WOIComposer
 jpip::WOIComposer, [186](#)
 Wait
 ipc::Event, [81](#)
 ipc::IPCObject, [118](#)
 ipc::Mutex, [123](#)
 ipc::RdWrLock, [148](#)
 WaitForInput
 net::Socket, [171](#)
 WaitForOutput
 net::Socket, [171](#)
 WaitForWriting
 ipc::RdWrLock, [148](#)
 WaitResult
 ipc, [14](#)
 woi
 jpip::DataBinServer, [69](#)
 woi_composer
 jpip::DataBinServer, [69](#)
 woi_position
 jpip::Request, [153](#)
 woi_size
 jpip::Request, [153](#)
 Write
 data::BaseFile, [34](#)
 jpip::DataBinWriter, [73](#)
 WriteByte
 data::BaseFile, [36](#)
 WriteEOR
 jpip::DataBinWriter, [75](#)
 WriteEmpty
 jpip::DataBinWriter, [73](#)
 WriteHeader
 jpip::DataBinWriter, [75](#)
 WritePlaceholder
 jpip::DataBinServer, [67](#)
 jpip::DataBinWriter, [75](#)
 WriteReverse
 data::BaseFile, [36](#)
 WriteSegment
 jpip::DataBinServer, [68](#)
 WriteVBAS
 jpip::DataBinWriter, [76](#)
 WriteValue
 jpip::DataBinWriter, [76](#)

 x
 jpeg2000::Point, [139](#)
 XML__BOX_ID
 file_manager.cc, [205](#)

 y
 jpeg2000::Point, [139](#)