

ZeroMQ Component Model

Generated by Doxygen 1.8.11

Contents

1	Deprecated List	1
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Namespace Documentation	11
6.1	Json Namespace Reference	11
6.1.1	Detailed Description	14
6.1.2	Typedef Documentation	14
6.1.2.1	ArrayIndex	14
6.1.2.2	CharReaderPtr	14
6.1.2.3	Int	14
6.1.2.4	Int64	14
6.1.2.5	LargestInt	14
6.1.2.6	LargestUInt	14
6.1.2.7	StreamWriterPtr	14
6.1.2.8	UInt	14

6.1.2.9	UInt64	14
6.1.2.10	UIntToStringBuffer	14
6.1.3	Enumeration Type Documentation	14
6.1.3.1	anonymous enum	14
6.1.3.2	CommentPlacement	14
6.1.3.3	ValueType	15
6.1.4	Function Documentation	15
6.1.4.1	ALIGNAS(8) kNull[sizeof(Value)]	15
6.1.4.2	codePointToUTF8(unsigned int cp)	15
6.1.4.3	containsControlCharacter(const char *str)	15
6.1.4.4	containsControlCharacter0(const char *str, unsigned len)	15
6.1.4.5	containsNewLine(Reader::Location begin, Reader::Location end)	15
6.1.4.6	decodePrefixedString(bool isPrefixed, char const *prefixed, unsigned *length, char const **value)	15
6.1.4.7	duplicateAndPrefixStringValue(const char *value, unsigned int length)	15
6.1.4.8	duplicateStringValue(const char *value, size_t length)	15
6.1.4.9	fixNumericLocale(char *begin, char *end)	16
6.1.4.10	getValidReaderKeys(std::set< JSONCPP_STRING > *valid_keys)	16
6.1.4.11	getValidWriterKeys(std::set< JSONCPP_STRING > *valid_keys)	16
6.1.4.12	InRange(double d, T min, U max)	16
6.1.4.13	isControlCharacter(char ch)	16
6.1.4.14	IsIntegral(double d)	16
6.1.4.15	normalizeEOL(Reader::Location begin, Reader::Location end)	16
6.1.4.16	operator<<(JSONCPP_OSTREAM &, const Value &root)	16
6.1.4.17	operator>>(JSONCPP_ISTREAM &, Value &)	16
6.1.4.18	parseFromStream(CharReader::Factory const &, JSONCPP_ISTREAM &, Value *root, std::string *errs)	17
6.1.4.19	parseFromStream(CharReader::Factory const &fact, JSONCPP_ISTREAM &sin, Value *root, JSONCPP_STRING *errs)	17
6.1.4.20	releasePrefixedStringValue(char *value)	17
6.1.4.21	releaseStringValue(char *value, unsigned)	17
6.1.4.22	strnpbrk(char const *s, char const *accept, size_t n)	17

6.1.4.23	throwLogicError(JSONCPP_STRING const &msg)	17
6.1.4.24	throwRuntimeError(JSONCPP_STRING const &msg)	17
6.1.4.25	uintToString(LargestUInt value, char *&current)	17
6.1.4.26	valueToQuotedString(const char *value)	18
6.1.4.27	valueToQuotedStringN(const char *value, unsigned length)	18
6.1.4.28	valueToString(Int value)	18
6.1.4.29	valueToString(UInt value)	18
6.1.4.30	valueToString(LargestInt value)	18
6.1.4.31	valueToString(LargestUInt value)	18
6.1.4.32	valueToString(double value, bool useSpecialFloats, unsigned int precision)	18
6.1.4.33	valueToString(double value)	18
6.1.4.34	valueToString(bool value)	18
6.1.4.35	writeString(StreamWriter::Factory const &factory, Value const &root)	18
6.1.5	Variable Documentation	18
6.1.5.1	kNullRef	18
6.2	std Namespace Reference	18
6.2.1	Function Documentation	18
6.2.1.1	swap(Json::Value &a, Json::Value &b)	18
6.3	zcm Namespace Reference	19
7	Class Documentation	21
7.1	zcm::Actor Class Reference	21
7.1.1	Detailed Description	21
7.1.2	Member Function Documentation	21
7.1.2.1	configure(std::string configuration_file)	21
7.1.2.2	get_name()	22
7.1.2.3	run()	22
7.1.3	Member Data Documentation	22
7.1.3.1	component_instances	22
7.1.3.2	name	22
7.2	zcm::Base_Operation Class Reference	22

7.2.1	Detailed Description	23
7.2.2	Constructor & Destructor Documentation	23
7.2.2.1	Base_Operation(std::string name, unsigned int priority)	23
7.2.3	Member Function Documentation	23
7.2.3.1	execute()	23
7.2.3.2	get_name()	23
7.2.3.3	get_priority() const	24
7.2.4	Member Data Documentation	24
7.2.4.1	name	24
7.2.4.2	priority	24
7.3	Json::BuiltStyledStreamWriter Struct Reference	24
7.3.1	Member Typedef Documentation	25
7.3.1.1	ChildValues	25
7.3.2	Constructor & Destructor Documentation	25
7.3.2.1	BuiltStyledStreamWriter(JSONCPP_STRING const &indentation, Comment← Style::Enum cs, JSONCPP_STRING const &colonSymbol, JSONCPP_STRING const &>nullSymbol, JSONCPP_STRING const &endingLineFeedSymbol, bool useSpecialFloats, unsigned int precision)	25
7.3.3	Member Function Documentation	25
7.3.3.1	hasCommentForValue(const Value &value)	26
7.3.3.2	indent()	26
7.3.3.3	isMultilineArray(Value const &value)	26
7.3.3.4	pushValue(JSONCPP_STRING const &value)	26
7.3.3.5	unindent()	26
7.3.3.6	write(Value const &root, JSONCPP_OSTREAM *sout) JSONCPP_OVERRIDE	26
7.3.3.7	writeArrayValue(Value const &value)	26
7.3.3.8	writeCommentAfterValueOnSameLine(Value const &root)	26
7.3.3.9	writeCommentBeforeValue(Value const &root)	26
7.3.3.10	writeIndent()	26
7.3.3.11	writeValue(Value const &value)	26
7.3.3.12	writeWithIndent(JSONCPP_STRING const &value)	26
7.3.4	Member Data Documentation	26

7.3.4.1	addChildValues_	26
7.3.4.2	childValues_	27
7.3.4.3	colonSymbol_	27
7.3.4.4	cs_	27
7.3.4.5	endingLineFeedSymbol_	27
7.3.4.6	indentation_	27
7.3.4.7	indented_	27
7.3.4.8	indentString_	27
7.3.4.9	nullSymbol_	27
7.3.4.10	precision_	27
7.3.4.11	rightMargin_	27
7.3.4.12	sout_	27
7.3.4.13	useSpecialFloats_	27
7.4	Json::CharReader Class Reference	27
7.4.1	Detailed Description	28
7.4.2	Constructor & Destructor Documentation	28
7.4.2.1	~CharReader()	28
7.4.3	Member Function Documentation	28
7.4.3.1	parse(char const *beginDoc, char const *endDoc, Value *root, JSONCPP_STRING *errs)=0	28
7.5	Json::CharReaderBuilder Class Reference	28
7.5.1	Detailed Description	29
7.5.2	Constructor & Destructor Documentation	29
7.5.2.1	CharReaderBuilder()	29
7.5.2.2	~CharReaderBuilder() JSONCPP_OVERRIDE	29
7.5.3	Member Function Documentation	29
7.5.3.1	newCharReader() const JSONCPP_OVERRIDE	29
7.5.3.2	operator[] (JSONCPP_STRING key)	30
7.5.3.3	setDefaults(Json::Value *settings)	30
7.5.3.4	strictMode(Json::Value *settings)	30
7.5.3.5	validate(Json::Value *invalid) const	31

7.5.4	Member Data Documentation	31
7.5.4.1	settings_	31
7.6	zcm::Client Class Reference	32
7.6.1	Detailed Description	32
7.6.2	Constructor & Destructor Documentation	32
7.6.2.1	Client(std::string name)	32
7.6.2.2	Client(std::string name, std::vector< std::string > endpoints)	33
7.6.2.3	~Client()	33
7.6.3	Member Function Documentation	33
7.6.3.1	call(std::string message)	33
7.6.3.2	connect(std::vector< std::string > new_endpoints)	33
7.6.3.3	get_name()	33
7.6.4	Member Data Documentation	33
7.6.4.1	client_socket	33
7.6.4.2	context	34
7.6.4.3	endpoints	34
7.6.4.4	name	34
7.7	Json::Value::CommentInfo Struct Reference	34
7.7.1	Constructor & Destructor Documentation	34
7.7.1.1	CommentInfo()	34
7.7.1.2	~CommentInfo()	34
7.7.2	Member Function Documentation	34
7.7.2.1	setComment(const char *text, size_t len)	34
7.7.3	Member Data Documentation	34
7.7.3.1	comment_	34
7.8	Json::CommentStyle Struct Reference	35
7.8.1	Detailed Description	35
7.8.2	Member Enumeration Documentation	35
7.8.2.1	Enum	35
7.9	zcm::Component Class Reference	35

7.9.1	Detailed Description	36
7.9.2	Constructor & Destructor Documentation	36
7.9.2.1	Component()	36
7.9.2.2	~Component()	36
7.9.3	Member Function Documentation	36
7.9.3.1	spawn()	36
7.9.4	Member Data Documentation	36
7.9.4.1	executor_thread	36
7.9.4.2	operation_queue	36
7.10	Json::Value::CZString Class Reference	37
7.10.1	Member Enumeration Documentation	37
7.10.1.1	DuplicationPolicy	37
7.10.2	Constructor & Destructor Documentation	38
7.10.2.1	CZString(ArrayIndex index)	38
7.10.2.2	CZString(char const *str, unsigned length, DuplicationPolicy allocate)	38
7.10.2.3	CZString(CZString const &other)	38
7.10.2.4	~CZString()	38
7.10.3	Member Function Documentation	38
7.10.3.1	data() const	38
7.10.3.2	index() const	38
7.10.3.3	isStaticString() const	38
7.10.3.4	length() const	38
7.10.3.5	operator<(CZString const &other) const	38
7.10.3.6	operator=(CZString other)	38
7.10.3.7	operator==(CZString const &other) const	38
7.10.3.8	swap(CZString &other)	38
7.10.4	Member Data Documentation	38
7.10.4.1	@2	38
7.10.4.2	cstr_	38
7.10.4.3	index_	38

7.10.4.4	storage_	38
7.11	Json::OurReader::ErrorInfo Class Reference	39
7.11.1	Member Data Documentation	39
7.11.1.1	extra_	39
7.11.1.2	message_	39
7.11.1.3	token_	39
7.12	Json::Reader::ErrorInfo Class Reference	39
7.12.1	Member Data Documentation	39
7.12.1.1	extra_	39
7.12.1.2	message_	39
7.12.1.3	token_	39
7.13	Json::Exception Class Reference	40
7.13.1	Detailed Description	40
7.13.2	Constructor & Destructor Documentation	40
7.13.2.1	Exception(JSONCPP_STRING const &msg)	40
7.13.2.2	~Exception() JSONCPP_OVERRIDE	40
7.13.3	Member Function Documentation	40
7.13.3.1	what() const JSONCPP_OVERRIDE	40
7.13.4	Member Data Documentation	40
7.13.4.1	msg_	40
7.14	Json::CharReader::Factory Class Reference	41
7.14.1	Constructor & Destructor Documentation	41
7.14.1.1	~Factory()	41
7.14.2	Member Function Documentation	41
7.14.2.1	newCharReader() const =0	41
7.15	Json::StreamWriter::Factory Class Reference	41
7.15.1	Detailed Description	42
7.15.2	Constructor & Destructor Documentation	42
7.15.2.1	~Factory()	42
7.15.3	Member Function Documentation	42

7.15.3.1	<code>newStreamWriter() const =0</code>	42
7.16	<code>Json::FastWriter</code> Class Reference	42
7.16.1	Detailed Description	43
7.16.2	Constructor & Destructor Documentation	43
7.16.2.1	<code>FastWriter()</code>	43
7.16.2.2	<code>~FastWriter() JSONCPP_OVERRIDE</code>	43
7.16.3	Member Function Documentation	43
7.16.3.1	<code>dropNullPlaceholders()</code>	43
7.16.3.2	<code>enableYAMLCompatibility()</code>	44
7.16.3.3	<code>omitEndingLineFeed()</code>	44
7.16.3.4	<code>write(const Value &root) JSONCPP_OVERRIDE</code>	44
7.16.3.5	<code>writeValue(const Value &value)</code>	44
7.16.4	Member Data Documentation	44
7.16.4.1	<code>document_</code>	44
7.16.4.2	<code>dropNullPlaceholders_</code>	44
7.16.4.3	<code>omitEndingLineFeed_</code>	44
7.16.4.4	<code>yamlCompatibilityEnabled_</code>	44
7.17	<code>Json::Features</code> Class Reference	44
7.17.1	Detailed Description	45
7.17.2	Constructor & Destructor Documentation	45
7.17.2.1	<code>Features()</code>	45
7.17.3	Member Function Documentation	45
7.17.3.1	<code>all()</code>	45
7.17.3.2	<code>strictMode()</code>	45
7.17.4	Member Data Documentation	46
7.17.4.1	<code>allowComments_</code>	46
7.17.4.2	<code>allowDroppedNullPlaceholders_</code>	46
7.17.4.3	<code>allowNumericKeys_</code>	46
7.17.4.4	<code>strictRoot_</code>	46
7.18	<code>Json::LogicError</code> Class Reference	46

7.18.1 Detailed Description	47
7.18.2 Constructor & Destructor Documentation	47
7.18.2.1 LogicError(JSONCPP_STRING const &msg)	47
7.18.3 Member Function Documentation	47
7.18.3.1 what() const JSONCPP_OVERRIDE	47
7.18.4 Member Data Documentation	47
7.18.4.1 msg_	47
7.19 zcm::Operation_Queue Class Reference	47
7.19.1 Detailed Description	48
7.19.2 Member Function Documentation	48
7.19.2.1 dequeue()	48
7.19.2.2 empty()	48
7.19.2.3 enqueue(Base_Operation *new_operation)	48
7.19.2.4 process()	48
7.19.2.5 spawn()	48
7.19.2.6 top()	48
7.19.3 Member Data Documentation	48
7.19.3.1 operation_queue	48
7.19.3.2 queue_mutex	49
7.20 Json::OurCharReader Class Reference	49
7.20.1 Constructor & Destructor Documentation	49
7.20.1.1 OurCharReader(bool collectComments, OurFeatures const &features)	49
7.20.2 Member Function Documentation	49
7.20.2.1 parse(char const *beginDoc, char const *endDoc, Value *root, JSONCPP_STRING *errs) JSONCPP_OVERRIDE	49
7.20.3 Member Data Documentation	50
7.20.3.1 collectComments_	50
7.20.3.2 reader_	50
7.21 Json::OurFeatures Class Reference	50
7.21.1 Member Function Documentation	51
7.21.1.1 all()	51

7.21.2	Member Data Documentation	51
7.21.2.1	allowComments_	51
7.21.2.2	allowDroppedNullPlaceholders_	51
7.21.2.3	allowNumericKeys_	51
7.21.2.4	allowSingleQuotes_	51
7.21.2.5	allowSpecialFloats_	51
7.21.2.6	failIfExtra_	51
7.21.2.7	rejectDupKeys_	51
7.21.2.8	stackLimit_	51
7.21.2.9	strictRoot_	51
7.22	Json::OurReader Class Reference	51
7.22.1	Member Typedef Documentation	53
7.22.1.1	Char	53
7.22.1.2	Errors	53
7.22.1.3	Location	53
7.22.1.4	Nodes	53
7.22.2	Member Enumeration Documentation	53
7.22.2.1	TokenType	53
7.22.3	Constructor & Destructor Documentation	54
7.22.3.1	OurReader(OurFeatures const &features)	54
7.22.3.2	OurReader(OurReader const &)	54
7.22.4	Member Function Documentation	54
7.22.4.1	addComment(Location begin, Location end, CommentPlacement placement)	54
7.22.4.2	addError(const JSONCPP_STRING &message, Token &token, Location extra=0)	54
7.22.4.3	addErrorAndRecover(const JSONCPP_STRING &message, Token &token, TokenType skipUntilToken)	54
7.22.4.4	currentValue()	54
7.22.4.5	decodeDouble(Token &token)	54
7.22.4.6	decodeDouble(Token &token, Value &decoded)	54
7.22.4.7	decodeNumber(Token &token)	54
7.22.4.8	decodeNumber(Token &token, Value &decoded)	54

7.22.4.9 decodeString(Token &token)	54
7.22.4.10 decodeString(Token &token, JSONCPP_STRING &decoded)	54
7.22.4.11 decodeUnicodeCodePoint(Token &token, Location ¤t, Location end, unsigned int &unicode)	54
7.22.4.12 decodeUnicodeEscapeSequence(Token &token, Location ¤t, Location end, unsigned int &unicode)	54
7.22.4.13 getFormattedErrorMessages() const	54
7.22.4.14 getLocationLineAndColumn(Location location, int &line, int &column) const	54
7.22.4.15 getLocationLineAndColumn(Location location) const	54
7.22.4.16 getNextChar()	54
7.22.4.17 getStructuredErrors() const	55
7.22.4.18 good() const	55
7.22.4.19 match(Location pattern, int patternLength)	55
7.22.4.20 operator=(OurReader const &)	55
7.22.4.21 parse(const char *beginDoc, const char *endDoc, Value &root, bool collect← Comments=true)	55
7.22.4.22 pushError(const Value &value, const JSONCPP_STRING &message)	55
7.22.4.23 pushError(const Value &value, const JSONCPP_STRING &message, const Value &extra)	55
7.22.4.24 readArray(Token &token)	55
7.22.4.25 readComment()	55
7.22.4.26 readCppStyleComment()	55
7.22.4.27 readCStyleComment()	55
7.22.4.28 readNumber(bool checkInf)	55
7.22.4.29 readObject(Token &token)	55
7.22.4.30 readString()	55
7.22.4.31 readStringSingleQuote()	55
7.22.4.32 readToken(Token &token)	55
7.22.4.33 readValue()	55
7.22.4.34 recoverFromError(TokenType skipUntilToken)	55
7.22.4.35 skipCommentTokens(Token &token)	55
7.22.4.36 skipSpaces()	55

7.22.4.37 skipUntilSpace()	55
7.22.5 Member Data Documentation	55
7.22.5.1 begin_	56
7.22.5.2 collectComments_	56
7.22.5.3 commentsBefore_	56
7.22.5.4 current_	56
7.22.5.5 document_	56
7.22.5.6 end_	56
7.22.5.7 errors_	56
7.22.5.8 features_	56
7.22.5.9 lastValue_	56
7.22.5.10 lastValueEnd_	56
7.22.5.11 nodes_	56
7.22.5.12 stackDepth_	56
7.23 Json::Path Class Reference	56
7.23.1 Detailed Description	57
7.23.2 Member Typedef Documentation	57
7.23.2.1 Args	57
7.23.2.2 InArgs	57
7.23.3 Constructor & Destructor Documentation	57
7.23.3.1 Path(const JSONCPP_STRING &path, const PathArgument &a1=PathArgument(), const PathArgument &a2=PathArgument(), const PathArgument &a3=PathArgument(), const PathArgument &a4=PathArgument(), const PathArgument &a5=PathArgument())	57
7.23.4 Member Function Documentation	57
7.23.4.1 addPathInArg(const JSONCPP_STRING &path, const InArgs &in, InArgs::const_iterator &itInArg, PathArgument::Kind kind)	57
7.23.4.2 invalidPath(const JSONCPP_STRING &path, int location)	57
7.23.4.3 make(Value &root) const	57
7.23.4.4 makePath(const JSONCPP_STRING &path, const InArgs &in)	58
7.23.4.5 resolve(const Value &root) const	58
7.23.4.6 resolve(const Value &root, const Value &defaultValue) const	58

7.23.5	Member Data Documentation	58
7.23.5.1	args_	58
7.24	Json::PathArgument Class Reference	58
7.24.1	Detailed Description	58
7.24.2	Member Enumeration Documentation	59
7.24.2.1	Kind	59
7.24.3	Constructor & Destructor Documentation	59
7.24.3.1	PathArgument()	59
7.24.3.2	PathArgument(ArrayIndex index)	59
7.24.3.3	PathArgument(const char *key)	59
7.24.3.4	PathArgument(const JSONCPP_STRING &key)	59
7.24.4	Friends And Related Function Documentation	59
7.24.4.1	Path	59
7.24.5	Member Data Documentation	59
7.24.5.1	index_	59
7.24.5.2	key_	59
7.24.5.3	kind_	59
7.25	zcm::Operation_Queue::PriorityOrdering Struct Reference	59
7.25.1	Member Function Documentation	60
7.25.1.1	operator()(const Base_Operation *lhs, const Base_Operation *rhs) const	60
7.26	zcm::Publisher Class Reference	60
7.26.1	Detailed Description	60
7.26.2	Constructor & Destructor Documentation	60
7.26.2.1	Publisher(std::string name)	60
7.26.2.2	Publisher(std::string name, std::vector< std::string > endpoints)	61
7.26.2.3	~Publisher()	61
7.26.3	Member Function Documentation	61
7.26.3.1	add_connection(std::string new_connection)	61
7.26.3.2	bind(std::vector< std::string > new_endpoints)	61
7.26.3.3	get_name()	61

7.26.3.4	send(std::string message)	62
7.26.4	Member Data Documentation	62
7.26.4.1	context	62
7.26.4.2	endpoints	62
7.26.4.3	name	62
7.26.4.4	publisher_socket	62
7.27	Json::Reader Class Reference	62
7.27.1	Detailed Description	64
7.27.2	Member Typedef Documentation	65
7.27.2.1	Char	65
7.27.2.2	Errors	65
7.27.2.3	Location	65
7.27.2.4	Nodes	65
7.27.3	Member Enumeration Documentation	65
7.27.3.1	TokenType	65
7.27.4	Constructor & Destructor Documentation	65
7.27.4.1	Reader()	65
7.27.4.2	Reader(const Features &features)	65
7.27.5	Member Function Documentation	66
7.27.5.1	addComment(Location begin, Location end, CommentPlacement placement)	66
7.27.5.2	addError(const JSONCPP_STRING &message, Token &token, Location extra=0)	66
7.27.5.3	addErrorAndRecover(const JSONCPP_STRING &message, Token &token, TokenType skipUntilToken)	66
7.27.5.4	currentValue()	66
7.27.5.5	decodeDouble(Token &token)	66
7.27.5.6	decodeDouble(Token &token, Value &decoded)	66
7.27.5.7	decodeNumber(Token &token)	66
7.27.5.8	decodeNumber(Token &token, Value &decoded)	66
7.27.5.9	decodeString(Token &token)	66
7.27.5.10	decodeString(Token &token, JSONCPP_STRING &decoded)	66

7.27.5.11 decodeUnicodeCodePoint(Token &token, Location ¤t, Location end, unsigned int &unicode)	66
7.27.5.12 decodeUnicodeEscapeSequence(Token &token, Location ¤t, Location end, unsigned int &unicode)	66
7.27.5.13 getFormattedErrorMessages() const	66
7.27.5.14 getFormattedErrorMessages() const	66
7.27.5.15 getLocationLineAndColumn(Location location, int &line, int &column) const . . .	67
7.27.5.16 getLocationLineAndColumn(Location location) const	67
7.27.5.17 getNextChar()	67
7.27.5.18 getStructuredErrors() const	67
7.27.5.19 good() const	67
7.27.5.20 match(Location pattern, int patternLength)	67
7.27.5.21 parse(const std::string &document, Value &root, bool collectComments=true) . .	67
7.27.5.22 parse(const char *beginDoc, const char *endDoc, Value &root, bool collect← Comments=true)	67
7.27.5.23 parse(JSONCPP_ISTREAM &is, Value &root, bool collectComments=true) . . .	68
7.27.5.24 pushError(const Value &value, const JSONCPP_STRING &message)	68
7.27.5.25 pushError(const Value &value, const JSONCPP_STRING &message, const Value &extra)	68
7.27.5.26 readArray(Token &token)	69
7.27.5.27 readComment()	69
7.27.5.28 readCppStyleComment()	69
7.27.5.29 readCStyleComment()	69
7.27.5.30 readNumber()	69
7.27.5.31 readObject(Token &token)	69
7.27.5.32 readString()	69
7.27.5.33 readToken(Token &token)	69
7.27.5.34 readValue()	69
7.27.5.35 recoverFromError(TokenType skipUntilToken)	69
7.27.5.36 skipCommentTokens(Token &token)	69
7.27.5.37 skipSpaces()	69
7.27.5.38 skipUntilSpace()	69

7.27.6	Member Data Documentation	69
7.27.6.1	begin_	69
7.27.6.2	collectComments_	69
7.27.6.3	commentsBefore_	69
7.27.6.4	current_	69
7.27.6.5	document_	69
7.27.6.6	end_	69
7.27.6.7	errors_	69
7.27.6.8	features_	69
7.27.6.9	lastValue_	69
7.27.6.10	lastValueEnd_	69
7.27.6.11	nodes_	69
7.28	Json::RuntimeError Class Reference	70
7.28.1	Detailed Description	70
7.28.2	Constructor & Destructor Documentation	70
7.28.2.1	RuntimeError(JSONCPP_STRING const &msg)	70
7.28.3	Member Function Documentation	70
7.28.3.1	what() const JSONCPP_OVERRIDE	70
7.28.4	Member Data Documentation	70
7.28.4.1	msg_	70
7.29	zcm::Server Class Reference	71
7.29.1	Detailed Description	72
7.29.2	Constructor & Destructor Documentation	72
7.29.2.1	Server(std::string name, unsigned int priority, std::function< std::string(const std::string &)> operation_function, Operation_Queue *operation_queue_ptr)	72
7.29.2.2	Server(std::string name, unsigned int priority, std::vector< std::string > endpoints, std::function< std::string(const std::string &)> operation_function, Operation_Queue *operation_queue_ptr)	72
7.29.2.3	~Server()	72
7.29.3	Member Function Documentation	72
7.29.3.1	add_connection(std::string new_connection)	72
7.29.3.2	bind(std::vector< std::string > new_endpoints)	73

7.29.3.3	<code>get_name()</code>	73
7.29.3.4	<code>get_priority()</code>	73
7.29.3.5	<code>rebind_operation_function(std::function< std::string(const std::string &)> new← _operation_function)</code>	73
7.29.3.6	<code>recv()</code>	73
7.29.3.7	<code>spawn()</code>	73
7.29.3.8	<code>start()</code>	73
7.29.4	Member Data Documentation	73
7.29.4.1	<code>context</code>	73
7.29.4.2	<code>endpoints</code>	74
7.29.4.3	<code>func_mutex</code>	74
7.29.4.4	<code>name</code>	74
7.29.4.5	<code>operation_function</code>	74
7.29.4.6	<code>operation_queue_ptr</code>	74
7.29.4.7	<code>priority</code>	74
7.29.4.8	<code>ready</code>	74
7.29.4.9	<code>server_socket</code>	74
7.30	<code>zcm::Server_Operation</code> Class Reference	75
7.30.1	Detailed Description	75
7.30.2	Constructor & Destructor Documentation	75
7.30.2.1	<code>Server_Operation(std::string name, unsigned int priority, std::function< std← ::string(> operation_function, zmq::socket_t *socket_ptr, bool *recv_ready)</code>	75
7.30.3	Member Function Documentation	76
7.30.3.1	<code>execute()</code>	76
7.30.3.2	<code>get_name()</code>	76
7.30.3.3	<code>get_priority() const</code>	76
7.30.3.4	<code>get_socket_ptr()</code>	76
7.30.3.5	<code>set_ready()</code>	76
7.30.4	Member Data Documentation	76
7.30.4.1	<code>operation_function</code>	76
7.30.4.2	<code>recv_ready</code>	77

7.30.4.3	<code>socket_ptr</code>	77
7.31	<code>Json::StaticString</code> Class Reference	77
7.31.1	Detailed Description	77
7.31.2	Constructor & Destructor Documentation	78
7.31.2.1	<code>StaticString(const char *cstring)</code>	78
7.31.3	Member Function Documentation	78
7.31.3.1	<code>c_str() const</code>	78
7.31.3.2	<code>operator const char *() const</code>	78
7.31.4	Member Data Documentation	78
7.31.4.1	<code>c_str_</code>	78
7.32	<code>Json::StreamWriter</code> Class Reference	78
7.32.1	Detailed Description	79
7.32.2	Constructor & Destructor Documentation	79
7.32.2.1	<code>StreamWriter()</code>	79
7.32.2.2	<code>~StreamWriter()</code>	79
7.32.3	Member Function Documentation	79
7.32.3.1	<code>write(Value const &root, JSONCPP_OSTREAM *sout)=0</code>	79
7.32.4	Member Data Documentation	79
7.32.4.1	<code>sout_</code>	80
7.33	<code>Json::StreamWriterBuilder</code> Class Reference	80
7.33.1	Detailed Description	80
7.33.2	Constructor & Destructor Documentation	81
7.33.2.1	<code>StreamWriterBuilder()</code>	81
7.33.2.2	<code>~StreamWriterBuilder() JSONCPP_OVERRIDE</code>	81
7.33.3	Member Function Documentation	81
7.33.3.1	<code>newStreamWriter() const JSONCPP_OVERRIDE</code>	81
7.33.3.2	<code>operator[](JSONCPP_STRING key)</code>	81
7.33.3.3	<code>setDefaults(Json::Value *settings)</code>	81
7.33.3.4	<code>validate(Json::Value *invalid) const</code>	81
7.33.4	Member Data Documentation	82

7.33.4.1 settings_	82
7.34 Json::Value::CZString::StringStorage Struct Reference	82
7.34.1 Member Data Documentation	82
7.34.1.1 length_	82
7.34.1.2 policy_	82
7.35 Json::Reader::StructuredError Struct Reference	83
7.35.1 Detailed Description	83
7.35.2 Member Data Documentation	83
7.35.2.1 message	83
7.35.2.2 offset_limit	83
7.35.2.3 offset_start	83
7.36 Json::OurReader::StructuredError Struct Reference	83
7.36.1 Member Data Documentation	83
7.36.1.1 message	83
7.36.1.2 offset_limit	83
7.36.1.3 offset_start	83
7.37 Json::StyledStreamWriter Class Reference	84
7.37.1 Detailed Description	85
7.37.2 Member Typedef Documentation	85
7.37.2.1 ChildValues	85
7.37.3 Constructor & Destructor Documentation	85
7.37.3.1 StyledStreamWriter(JSONCPP_STRING indentation=""\t"")	85
7.37.3.2 ~StyledStreamWriter()	85
7.37.4 Member Function Documentation	85
7.37.4.1 hasCommentForValue(const Value &value)	85
7.37.4.2 indent()	85
7.37.4.3 isMultilineArray(const Value &value)	86
7.37.4.4 normalizeEOL(const JSONCPP_STRING &text)	86
7.37.4.5 pushValue(const JSONCPP_STRING &value)	86
7.37.4.6 unindent()	86

7.37.4.7	write(JSONCPP_OSTREAM &out, const Value &root)	86
7.37.4.8	writeArrayValue(const Value &value)	86
7.37.4.9	writeCommentAfterValueOnSameLine(const Value &root)	86
7.37.4.10	writeCommentBeforeValue(const Value &root)	86
7.37.4.11	writeIndent()	86
7.37.4.12	writeValue(const Value &value)	86
7.37.4.13	writeWithIndent(const JSONCPP_STRING &value)	86
7.37.5	Member Data Documentation	86
7.37.5.1	addChildValues_	86
7.37.5.2	childValues_	86
7.37.5.3	document_	86
7.37.5.4	indentation_	86
7.37.5.5	indented_	86
7.37.5.6	indentString_	86
7.37.5.7	rightMargin_	86
7.38	Json::StyledWriter Class Reference	87
7.38.1	Detailed Description	88
7.38.2	Member Typedef Documentation	88
7.38.2.1	ChildValues	88
7.38.3	Constructor & Destructor Documentation	88
7.38.3.1	StyledWriter()	88
7.38.3.2	~StyledWriter() JSONCPP_OVERRIDE	88
7.38.4	Member Function Documentation	88
7.38.4.1	hasCommentForValue(const Value &value)	88
7.38.4.2	indent()	88
7.38.4.3	isMultilineArray(const Value &value)	88
7.38.4.4	normalizeEOL(const JSONCPP_STRING &text)	88
7.38.4.5	pushValue(const JSONCPP_STRING &value)	88
7.38.4.6	unindent()	88
7.38.4.7	write(const Value &root) JSONCPP_OVERRIDE	88

7.38.4.8	<code>writeArrayValue(const Value &value)</code>	89
7.38.4.9	<code>writeCommentAfterValueOnSameLine(const Value &root)</code>	89
7.38.4.10	<code>writeCommentBeforeValue(const Value &root)</code>	89
7.38.4.11	<code>writeIndent()</code>	89
7.38.4.12	<code>writeValue(const Value &value)</code>	89
7.38.4.13	<code>writeWithIndent(const JSONCPP_STRING &value)</code>	89
7.38.5	Member Data Documentation	89
7.38.5.1	<code>addChildValues_</code>	89
7.38.5.2	<code>childValues_</code>	89
7.38.5.3	<code>document_</code>	89
7.38.5.4	<code>indentSize_</code>	89
7.38.5.5	<code>indentString_</code>	89
7.38.5.6	<code>rightMargin_</code>	89
7.39	<code>zcm::Subscriber</code> Class Reference	89
7.39.1	Detailed Description	91
7.39.2	Constructor & Destructor Documentation	91
7.39.2.1	<code>Subscriber(std::string name, unsigned int priority, std::string filter, std::function< void(const std::string &)> operation_function, Operation_Queue *operation_queue_ptr)</code>	91
7.39.2.2	<code>Subscriber(std::string name, unsigned int priority, std::string filter, std::vector< std::string > endpoints, std::function< void(const std::string &)> operation_function, Operation_Queue *operation_queue_ptr)</code>	91
7.39.2.3	<code>~Subscriber()</code>	91
7.39.3	Member Function Documentation	91
7.39.3.1	<code>add_connection(std::string new_connection)</code>	91
7.39.3.2	<code>connect(std::vector< std::string > new_endpoints)</code>	92
7.39.3.3	<code>get_name()</code>	92
7.39.3.4	<code>get_priority()</code>	92
7.39.3.5	<code>rebind_operation_function(std::function< void(const std::string &)> new_operation_function)</code>	92
7.39.3.6	<code>recv()</code>	92
7.39.3.7	<code>spawn()</code>	92
7.39.3.8	<code>start()</code>	92

7.39.4	Member Data Documentation	93
7.39.4.1	context	93
7.39.4.2	endpoints	93
7.39.4.3	filter	93
7.39.4.4	func_mutex	93
7.39.4.5	name	93
7.39.4.6	operation_function	93
7.39.4.7	operation_queue_ptr	93
7.39.4.8	priority	93
7.39.4.9	subscriber_socket	93
7.40	zcm::Subscriber_Operation Class Reference	94
7.40.1	Detailed Description	94
7.40.2	Constructor & Destructor Documentation	94
7.40.2.1	Subscriber_Operation(std::string name, unsigned int priority, std::function< void()> operation_function)	94
7.40.3	Member Function Documentation	95
7.40.3.1	execute()	95
7.40.3.2	get_name()	95
7.40.3.3	get_priority() const	95
7.40.4	Member Data Documentation	95
7.40.4.1	operation_function	95
7.41	zcm::Timer Class Reference	95
7.41.1	Detailed Description	96
7.41.2	Constructor & Destructor Documentation	96
7.41.2.1	Timer(std::string name, unsigned int priority, long long period, std::function< void()> operation_function, Operation_Queue *operation_queue_ptr)	96
7.41.3	Member Function Documentation	97
7.41.3.1	change_period(long long new_period)	97
7.41.3.2	get_name()	97
7.41.3.3	get_priority()	97
7.41.3.4	operation()	97

7.41.3.5	rebind_operation_function(std::function< void()> new_operation_function)	97
7.41.3.6	spawn()	98
7.41.3.7	start()	98
7.41.4	Member Data Documentation	98
7.41.4.1	func_mutex	98
7.41.4.2	name	98
7.41.4.3	operation_function	98
7.41.4.4	operation_queue_ptr	98
7.41.4.5	period	98
7.41.4.6	period_mutex	98
7.41.4.7	priority	99
7.42	zcm::Timer_Operation Class Reference	99
7.42.1	Detailed Description	99
7.42.2	Constructor & Destructor Documentation	99
7.42.2.1	Timer_Operation(std::string name, unsigned int priority, std::function< void()> operation_function)	99
7.42.3	Member Function Documentation	100
7.42.3.1	execute()	100
7.42.3.2	get_name()	100
7.42.3.3	get_priority() const	100
7.42.4	Member Data Documentation	100
7.42.4.1	operation_function	100
7.43	Json::OurReader::Token Class Reference	101
7.43.1	Member Data Documentation	101
7.43.1.1	end_	101
7.43.1.2	start_	101
7.43.1.3	type_	101
7.44	Json::Reader::Token Class Reference	101
7.44.1	Member Data Documentation	101
7.44.1.1	end_	101
7.44.1.2	start_	101

7.44.1.3	<code>type_</code>	101
7.45	<code>Json::Value</code> Class Reference	101
7.45.1	Detailed Description	106
7.45.2	Member Typedef Documentation	106
7.45.2.1	<code>ArrayIndex</code>	106
7.45.2.2	<code>const_iterator</code>	106
7.45.2.3	<code>Int</code>	106
7.45.2.4	<code>Int64</code>	106
7.45.2.5	<code>iterator</code>	106
7.45.2.6	<code>LargestInt</code>	107
7.45.2.7	<code>LargestUInt</code>	107
7.45.2.8	Members	107
7.45.2.9	<code>ObjectValues</code>	107
7.45.2.10	<code>UInt</code>	107
7.45.2.11	<code>UInt64</code>	107
7.45.3	Constructor & Destructor Documentation	107
7.45.3.1	<code>Value(ValueType type=nullValue)</code>	107
7.45.3.2	<code>Value(Int value)</code>	107
7.45.3.3	<code>Value(UInt value)</code>	107
7.45.3.4	<code>Value(Int64 value)</code>	107
7.45.3.5	<code>Value(UInt64 value)</code>	107
7.45.3.6	<code>Value(double value)</code>	107
7.45.3.7	<code>Value(const char *value)</code>	107
7.45.3.8	<code>Value(const char *begin, const char *end)</code>	107
7.45.3.9	<code>Value(const StaticString &value)</code>	108
7.45.3.10	<code>Value(const JSONCPP_STRING &value)</code>	108
7.45.3.11	<code>Value(bool value)</code>	108
7.45.3.12	<code>Value(const Value &other)</code>	108
7.45.3.13	<code>~Value()</code>	108
7.45.4	Member Function Documentation	108

7.45.4.1	<code>append(const Value &value)</code>	108
7.45.4.2	<code>asBool() const</code>	108
7.45.4.3	<code>asCString() const</code>	108
7.45.4.4	<code>asDouble() const</code>	109
7.45.4.5	<code>asFloat() const</code>	109
7.45.4.6	<code>asInt() const</code>	109
7.45.4.7	<code>asInt64() const</code>	109
7.45.4.8	<code>asLargestInt() const</code>	109
7.45.4.9	<code>asLargestUInt() const</code>	109
7.45.4.10	<code>asString() const</code>	109
7.45.4.11	<code>asUInt() const</code>	109
7.45.4.12	<code>asUInt64() const</code>	109
7.45.4.13	<code>begin() const</code>	109
7.45.4.14	<code>begin()</code>	109
7.45.4.15	<code>clear()</code>	109
7.45.4.16	<code>compare(const Value &other) const</code>	109
7.45.4.17	<code>demand(char const *begin, char const *end)</code>	109
7.45.4.18	<code>empty() const</code>	110
7.45.4.19	<code>end() const</code>	110
7.45.4.20	<code>end()</code>	110
7.45.4.21	<code>find(char const *begin, char const *end) const</code>	110
7.45.4.22	<code>get(ArrayIndex index, const Value &defaultValue) const</code>	110
7.45.4.23	<code>get(const char *key, const Value &defaultValue) const</code>	110
7.45.4.24	<code>get(const char *begin, const char *end, const Value &defaultValue) const</code>	110
7.45.4.25	<code>get(const JSONCPP_STRING &key, const Value &defaultValue) const</code>	110
7.45.4.26	<code>getComment(CommentPlacement placement) const</code>	111
7.45.4.27	<code>getMemberNames() const</code>	111
7.45.4.28	<code>getOffsetLimit() const</code>	111
7.45.4.29	<code>getOffsetStart() const</code>	111
7.45.4.30	<code>getString(char const **begin, char const **end) const</code>	111

7.45.4.31 hasComment(CommentPlacement placement) const	112
7.45.4.32 initBasic(ValueType type, bool allocated=false)	112
7.45.4.33 isArray() const	112
7.45.4.34 isBool() const	112
7.45.4.35 isConvertibleTo(ValueType other) const	112
7.45.4.36 isDouble() const	112
7.45.4.37 isInt() const	112
7.45.4.38 isInt64() const	112
7.45.4.39 isIntegral() const	112
7.45.4.40 isMember(const char *key) const	112
7.45.4.41 isMember(const JSONCPP_STRING &key) const	112
7.45.4.42 isMember(const char *begin, const char *end) const	112
7.45.4.43 isNull() const	112
7.45.4.44 isNumeric() const	112
7.45.4.45 isObject() const	112
7.45.4.46 isString() const	113
7.45.4.47 isUInt() const	113
7.45.4.48 isUInt64() const	113
7.45.4.49 isValidIndex(ArrayIndex index) const	113
7.45.4.50 operator"()() const	113
7.45.4.51 operator"!=(const Value &other) const	113
7.45.4.52 operator<(const Value &other) const	113
7.45.4.53 operator<=(const Value &other) const	113
7.45.4.54 operator=(Value other)	113
7.45.4.55 operator==(const Value &other) const	113
7.45.4.56 operator>(const Value &other) const	113
7.45.4.57 operator>=(const Value &other) const	113
7.45.4.58 operator[](ArrayIndex index)	113
7.45.4.59 operator[](int index)	114
7.45.4.60 operator[](ArrayIndex index) const	114

7.45.4.61	<code>operator[](int index) const</code>	114
7.45.4.62	<code>operator[](const char *key)</code>	114
7.45.4.63	<code>operator[](const char *key) const</code>	114
7.45.4.64	<code>operator[](const JSONCPP_STRING &key)</code>	114
7.45.4.65	<code>operator[](const JSONCPP_STRING &key) const</code>	114
7.45.4.66	<code>operator[](const StaticString &key)</code>	115
7.45.4.67	<code>removeIndex(ArrayIndex i, Value *removed)</code>	115
7.45.4.68	<code>removeMember(const char *key)</code>	115
7.45.4.69	<code>removeMember(const JSONCPP_STRING &key)</code>	115
7.45.4.70	<code>removeMember(const char *key, Value *removed)</code>	116
7.45.4.71	<code>removeMember(JSONCPP_STRING const &key, Value *removed)</code>	116
7.45.4.72	<code>removeMember(const char *begin, const char *end, Value *removed)</code>	116
7.45.4.73	<code>resize(ArrayIndex size)</code>	116
7.45.4.74	<code>resolveReference(const char *key)</code>	117
7.45.4.75	<code>resolveReference(const char *key, const char *end)</code>	117
7.45.4.76	<code>setComment(const char *comment, CommentPlacement placement)</code>	117
7.45.4.77	<code>setComment(const char *comment, size_t len, CommentPlacement placement)</code>	117
7.45.4.78	<code>setComment(const JSONCPP_STRING &comment, CommentPlacement placement)</code>	117
7.45.4.79	<code>setOffsetLimit(ptrdiff_t limit)</code>	117
7.45.4.80	<code>setOffsetStart(ptrdiff_t start)</code>	117
7.45.4.81	<code>size() const</code>	117
7.45.4.82	<code>swap(Value &other)</code>	117
7.45.4.83	<code>swapPayload(Value &other)</code>	117
7.45.4.84	<code>toStyledString() const</code>	117
7.45.4.85	<code>type() const</code>	117
7.45.5	Friends And Related Function Documentation	117
7.45.5.1	<code>ValueIteratorBase</code>	117
7.45.6	Member Data Documentation	117
7.45.6.1	<code>allocated_</code>	117
7.45.6.2	<code>comments_</code>	117

7.45.6.3	limit_	117
7.45.6.4	maxInt	117
7.45.6.5	maxInt64	118
7.45.6.6	maxLargestInt	118
7.45.6.7	maxLargestUInt	118
7.45.6.8	maxUInt	118
7.45.6.9	maxUInt64	118
7.45.6.10	minInt	118
7.45.6.11	minInt64	118
7.45.6.12	minLargestInt	118
7.45.6.13	null	118
7.45.6.14	nullRef	118
7.45.6.15	start_	119
7.45.6.16	type_	119
7.45.6.17	value_	119
7.46	Json::ValueConstIterator Class Reference	119
7.46.1	Detailed Description	120
7.46.2	Member Typedef Documentation	120
7.46.2.1	difference_type	120
7.46.2.2	iterator_category	120
7.46.2.3	pointer	120
7.46.2.4	reference	120
7.46.2.5	SelfType	120
7.46.2.6	size_t	120
7.46.2.7	value_type	121
7.46.3	Constructor & Destructor Documentation	121
7.46.3.1	ValueConstIterator()	121
7.46.3.2	ValueConstIterator(ValueIterator const &other)	121
7.46.3.3	ValueConstIterator(const Value::ObjectValues::iterator &current)	121
7.46.4	Member Function Documentation	121

7.46.4.1	<code>computeDistance(const SelfType &other) const</code>	121
7.46.4.2	<code>copy(const SelfType &other)</code>	121
7.46.4.3	<code>decrement()</code>	121
7.46.4.4	<code>deref() const</code>	121
7.46.4.5	<code>increment()</code>	121
7.46.4.6	<code>index() const</code>	121
7.46.4.7	<code>isEqual(const SelfType &other) const</code>	121
7.46.4.8	<code>key() const</code>	121
7.46.4.9	<code>memberName() const</code>	121
7.46.4.10	<code>memberName(char const **end) const</code>	122
7.46.4.11	<code>name() const</code>	122
7.46.4.12	<code>operator!=(const SelfType &other) const</code>	122
7.46.4.13	<code>operator*() const</code>	122
7.46.4.14	<code>operator++(int)</code>	122
7.46.4.15	<code>operator++()</code>	122
7.46.4.16	<code>operator-(const SelfType &other) const</code>	122
7.46.4.17	<code>operator--(int)</code>	122
7.46.4.18	<code>operator--()</code>	122
7.46.4.19	<code>operator->() const</code>	122
7.46.4.20	<code>operator=(const ValueIteratorBase &other)</code>	122
7.46.4.21	<code>operator==(const SelfType &other) const</code>	122
7.46.5	Friends And Related Function Documentation	122
7.46.5.1	Value	122
7.47	Json::Value::ValueHolder Union Reference	123
7.47.1	Member Data Documentation	123
7.47.1.1	<code>bool_</code>	123
7.47.1.2	<code>int_</code>	123
7.47.1.3	<code>map_</code>	123
7.47.1.4	<code>real_</code>	123
7.47.1.5	<code>string_</code>	123

7.47.1.6	<code>uint_</code>	123
7.48	<code>Json::ValueIterator</code> Class Reference	123
7.48.1	Detailed Description	125
7.48.2	Member Typedef Documentation	125
7.48.2.1	<code>difference_type</code>	125
7.48.2.2	<code>iterator_category</code>	125
7.48.2.3	<code>pointer</code>	125
7.48.2.4	<code>reference</code>	125
7.48.2.5	<code>SelfType</code>	125
7.48.2.6	<code>size_t</code>	125
7.48.2.7	<code>value_type</code>	125
7.48.3	Constructor & Destructor Documentation	125
7.48.3.1	<code>ValueIterator()</code>	125
7.48.3.2	<code>ValueIterator(const ValueConstIterator &other)</code>	125
7.48.3.3	<code>ValueIterator(const ValueIterator &other)</code>	125
7.48.3.4	<code>ValueIterator(const Value::ObjectValues::iterator &current)</code>	125
7.48.4	Member Function Documentation	125
7.48.4.1	<code>computeDistance(const SelfType &other) const</code>	125
7.48.4.2	<code>copy(const SelfType &other)</code>	125
7.48.4.3	<code>decrement()</code>	125
7.48.4.4	<code>deref() const</code>	125
7.48.4.5	<code>increment()</code>	125
7.48.4.6	<code>index() const</code>	125
7.48.4.7	<code>isEqual(const SelfType &other) const</code>	126
7.48.4.8	<code>key() const</code>	126
7.48.4.9	<code>memberName() const</code>	126
7.48.4.10	<code>memberName(char const **end) const</code>	126
7.48.4.11	<code>name() const</code>	126
7.48.4.12	<code>operator!=(const SelfType &other) const</code>	126
7.48.4.13	<code>operator*() const</code>	126

7.48.4.14	<code>operator++(int)</code>	126
7.48.4.15	<code>operator++()</code>	126
7.48.4.16	<code>operator-(const SelfType &other) const</code>	126
7.48.4.17	<code>operator--(int)</code>	126
7.48.4.18	<code>operator--()</code>	126
7.48.4.19	<code>operator->() const</code>	126
7.48.4.20	<code>operator=(const SelfType &other)</code>	126
7.48.4.21	<code>operator==(const SelfType &other) const</code>	126
7.48.5	Friends And Related Function Documentation	126
7.48.5.1	Value	126
7.49	Json::ValueIteratorBase Class Reference	127
7.49.1	Detailed Description	128
7.49.2	Member Typedef Documentation	128
7.49.2.1	<code>difference_type</code>	128
7.49.2.2	<code>iterator_category</code>	128
7.49.2.3	<code>SelfType</code>	128
7.49.2.4	<code>size_t</code>	128
7.49.3	Constructor & Destructor Documentation	128
7.49.3.1	<code>ValueIteratorBase()</code>	128
7.49.3.2	<code>ValueIteratorBase(const Value::ObjectValues::iterator &current)</code>	128
7.49.4	Member Function Documentation	128
7.49.4.1	<code>computeDistance(const SelfType &other) const</code>	128
7.49.4.2	<code>copy(const SelfType &other)</code>	128
7.49.4.3	<code>decrement()</code>	128
7.49.4.4	<code>deref() const</code>	128
7.49.4.5	<code>increment()</code>	128
7.49.4.6	<code>index() const</code>	128
7.49.4.7	<code>isEqual(const SelfType &other) const</code>	129
7.49.4.8	<code>key() const</code>	129
7.49.4.9	<code>memberName() const</code>	129

7.49.4.10	<code>memberName(char const **end) const</code>	129
7.49.4.11	<code>name() const</code>	129
7.49.4.12	<code>operator!=(const SelfType &other) const</code>	129
7.49.4.13	<code>operator-(const SelfType &other) const</code>	129
7.49.4.14	<code>operator==(const SelfType &other) const</code>	129
7.49.5	Member Data Documentation	129
7.49.5.1	<code>current_</code>	129
7.49.5.2	<code>isNull_</code>	129
7.50	Json::Writer Class Reference	130
7.50.1	Detailed Description	130
7.50.2	Constructor & Destructor Documentation	130
7.50.2.1	<code>~Writer()</code>	130
7.50.3	Member Function Documentation	130
7.50.3.1	<code>write(const Value &root)=0</code>	130
8	File Documentation	131
8.1	<code>/home/pranav/Repositories/zcm/include/actor.hpp</code> File Reference	131
8.1.1	Detailed Description	131
8.2	<code>/home/pranav/Repositories/zcm/include/client.hpp</code> File Reference	132
8.2.1	Detailed Description	132
8.3	<code>/home/pranav/Repositories/zcm/include/component.hpp</code> File Reference	132
8.3.1	Detailed Description	133
8.4	<code>/home/pranav/Repositories/zcm/include/json.hpp</code> File Reference	133
8.4.1	Macro Definition Documentation	136
8.4.1.1	<code>CPPTL_JSON_ASSERTIONS_H_INCLUDED</code>	136
8.4.1.2	<code>CPPTL_JSON_FEATURES_H_INCLUDED</code>	136
8.4.1.3	<code>CPPTL_JSON_H_INCLUDED</code>	136
8.4.1.4	<code>CPPTL_JSON_READER_H_INCLUDED</code>	136
8.4.1.5	<code>JSON_API</code>	136
8.4.1.6	<code>JSON_ASSERT</code>	137
8.4.1.7	<code>JSON_ASSERT_MESSAGE</code>	137

8.4.1.8	JSON_CONFIG_H_INCLUDED	137
8.4.1.9	JSON_FAIL_MESSAGE	137
8.4.1.10	JSON_FORWARDS_H_INCLUDED	137
8.4.1.11	JSON_HAS_INT64	137
8.4.1.12	JSON_HAS_RVALUE_REFERENCES	137
8.4.1.13	JSON_IS_AMALGAMATION	137
8.4.1.14	JSON_USE_EXCEPTION	137
8.4.1.15	JSON_VERSION_H_INCLUDED	138
8.4.1.16	JSON_WRITER_H_INCLUDED	138
8.4.1.17	JSONCPP_DEPRECATED	138
8.4.1.18	JSONCPP_ISTREAM	138
8.4.1.19	JSONCPP_ISTRINGSTREAM	138
8.4.1.20	JSONCPP_NORETURN	138
8.4.1.21	JSONCPP_OSTREAM	138
8.4.1.22	JSONCPP_OSTRINGSTREAM	138
8.4.1.23	JSONCPP_OVERRIDE	138
8.4.1.24	JSONCPP_STRING	138
8.4.1.25	JSONCPP_USING_SECURE_MEMORY	138
8.4.1.26	JSONCPP_VERSION_HEX	138
8.4.1.27	JSONCPP_VERSION_MAJOR	138
8.4.1.28	JSONCPP_VERSION_MINOR	138
8.4.1.29	JSONCPP_VERSION_PATCH	138
8.4.1.30	JSONCPP_VERSION_QUALIFIER	138
8.4.1.31	JSONCPP_VERSION_STRING	138
8.5	/home/pranav/Repositories/zcm/include/operation_queue.hpp File Reference	138
8.5.1	Detailed Description	139
8.6	/home/pranav/Repositories/zcm/include/operation_types.hpp File Reference	139
8.6.1	Detailed Description	140
8.7	/home/pranav/Repositories/zcm/include/publisher.hpp File Reference	140
8.7.1	Detailed Description	140

8.8	/home/pranav/Repositories/zcm/include/server.hpp File Reference	140
8.8.1	Detailed Description	141
8.9	/home/pranav/Repositories/zcm/include/subscriber.hpp File Reference	141
8.9.1	Detailed Description	142
8.10	/home/pranav/Repositories/zcm/include/timer.hpp File Reference	142
8.10.1	Detailed Description	142
8.11	/home/pranav/Repositories/zcm/src/actor.cpp File Reference	143
8.11.1	Detailed Description	143
8.12	/home/pranav/Repositories/zcm/src/client.cpp File Reference	143
8.12.1	Detailed Description	143
8.13	/home/pranav/Repositories/zcm/src/component.cpp File Reference	144
8.13.1	Detailed Description	144
8.14	/home/pranav/Repositories/zcm/src/json.cpp File Reference	144
8.14.1	Macro Definition Documentation	146
8.14.1.1	ALIGNAS	146
8.14.1.2	isfinite	146
8.14.1.3	JSON_ASSERT_UNREACHABLE	146
8.14.1.4	LIB_JSONCPP_JSON_TOOL_H_INCLUDED	146
8.14.2	Variable Documentation	147
8.14.2.1	stackDepth_g	147
8.14.2.2	stackLimit_g	147
8.15	/home/pranav/Repositories/zcm/src/operation_queue.cpp File Reference	147
8.15.1	Detailed Description	147
8.16	/home/pranav/Repositories/zcm/src/operation_types.cpp File Reference	147
8.16.1	Detailed Description	148
8.17	/home/pranav/Repositories/zcm/src/publisher.cpp File Reference	148
8.17.1	Detailed Description	148
8.18	/home/pranav/Repositories/zcm/src/server.cpp File Reference	148
8.18.1	Detailed Description	149
8.19	/home/pranav/Repositories/zcm/src/subscriber.cpp File Reference	149
8.19.1	Detailed Description	149
8.20	/home/pranav/Repositories/zcm/src/timer.cpp File Reference	149
8.20.1	Detailed Description	149

Chapter 1

Deprecated List

Class `Json::FastWriter`

Use `StreamWriterBuilder`.

Class `Json::Reader`

Use `CharReader` and `CharReaderBuilder`.

Member `Json::Reader::getFormattedErrorMessages ()` const

Use `getFormattedErrorMessages()` instead (typo fix).

Class `Json::StyledStreamWriter`

Use `StreamWriterBuilder`.

Class `Json::StyledWriter`

Use `StreamWriterBuilder`.

Member `Json::Value::removeMember` (const JSONCPP_STRING &key)

Member `Json::Value::removeMember` (const char *key)

Member `Json::Value::setComment` (const char *comment, CommentPlacement placement)

Always pass len.

Member `Json::ValueIteratorBase::memberName ()` const

This cannot be used for UTF-8 strings, since there can be embedded nulls.

Class `Json::Writer`

Use `StreamWriter`. (And really, this is an implementation detail.)

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Json	JSON (JavaScript Object Notation)	11
std	18
zcm	19

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

zcm::Actor	21
zcm::Base_Operation	22
zcm::Server_Operation	75
zcm::Subscriber_Operation	94
zcm::Timer_Operation	99
Json::CharReader	27
Json::OurCharReader	49
zcm::Client	32
Json::Value::CommentInfo	34
Json::CommentStyle	35
zcm::Component	35
Json::Value::CZString	37
Json::OurReader::ErrorInfo	39
Json::Reader::ErrorInfo	39
exception	
Json::Exception	40
Json::LogicError	46
Json::RuntimeError	70
Json::CharReader::Factory	41
Json::CharReaderBuilder	28
Json::StreamWriter::Factory	41
Json::StreamWriterBuilder	80
Json::Features	44
zcm::Operation_Queue	47
Json::OurFeatures	50
Json::OurReader	51
Json::Path	56
Json::PathArgument	58
zcm::Operation_Queue::PriorityOrdering	59
zcm::Publisher	60
Json::Reader	62
zcm::Server	71
Json::StaticString	77
Json::StreamWriter	78

Json::BuiltStyledStreamWriter	24
Json::Value::CZString::StringStorage	82
Json::Reader::StructuredError	83
Json::OurReader::StructuredError	83
Json::StyledStreamWriter	84
zcm::Subscriber	89
zcm::Timer	95
Json::OurReader::Token	101
Json::Reader::Token	101
Json::Value	101
Json::Value::ValueHolder	123
Json::ValueIteratorBase	127
Json::ValueConstIterator	119
Json::ValueIterator	123
Json::Writer	130
Json::FastWriter	42
Json::StyledWriter	87

Chapter 4

Class Index

4.1 Class List

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

zcm::Actor	
Actor class	21
zcm::Base_Operation	
Base Operation class	22
Json::BuiltStyledStreamWriter	24
Json::CharReader	
Interface for reading JSON from a char array	27
Json::CharReaderBuilder	
Build a CharReader implementation	28
zcm::Client	
Client class	32
Json::Value::CommentInfo	34
Json::CommentStyle	
Scoped enums are not available until C++11	35
zcm::Component	
Component class	35
Json::Value::CZString	37
Json::OurReader::ErrorInfo	39
Json::Reader::ErrorInfo	39
Json::Exception	
Base class for all exceptions we throw	40
Json::CharReader::Factory	41
Json::StreamWriter::Factory	
A simple abstract factory	41
Json::FastWriter	
Outputs a Value in JSON format without formatting (not human friendly)	42
Json::Features	
Configuration passed to reader and writer	44
Json::LogicError	
Exceptions thrown by JSON_ASSERT/JSON_FAIL macros	46
zcm::Operation_Queue	
Operation_Queue class	47
Json::OurCharReader	49
Json::OurFeatures	50
Json::OurReader	51

Json::Path	
Experimental and untested: represents a "path" to access a node	56
Json::PathArgument	
Experimental and untested: represents an element of the "path" to access a node	58
zcm::Operation_Queue::PriorityOrdering	59
zcm::Publisher	
Publisher class	60
Json::Reader	
Unserialize a JSON document into a Value	62
Json::RuntimeError	
Exceptions which the user cannot easily avoid	70
zcm::Server	
Server class	71
zcm::Server_Operation	
Server Operation class	75
Json::StaticString	
Lightweight wrapper to tag static string	77
Json::StreamWriter	
Usage:	78
Json::StreamWriterBuilder	
Build a StreamWriter implementation	80
Json::Value::CZString::StringStorage	82
Json::Reader::StructuredError	
An error tagged with where in the JSON text it was encountered	83
Json::OurReader::StructuredError	83
Json::StyledStreamWriter	
Writes a Value in JSON format in a human friendly way, to a stream rather than to a string . . .	84
Json::StyledWriter	
Writes a Value in JSON format in a human friendly way	87
zcm::Subscriber	
Subscriber class	89
zcm::Subscriber_Operation	
Subscriber Operation class	94
zcm::Timer	
Timer class	95
zcm::Timer_Operation	
Timer Operation class	99
Json::OurReader::Token	101
Json::Reader::Token	101
Json::Value	
Represents a JSON value	101
Json::ValueConstIterator	
Const iterator for object and array value	119
Json::Value::ValueHolder	123
Json::ValueIterator	
Iterator for object and array value	123
Json::ValueIteratorBase	
Base class for Value iterators	127
Json::Writer	
Abstract class for writers	130

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

/home/pranav/Repositories/zcm/include/ actor.hpp	131
This file declares the Actor class	
/home/pranav/Repositories/zcm/include/ client.hpp	132
This file declares the Client class	
/home/pranav/Repositories/zcm/include/ component.hpp	132
This file declares the Component class	
/home/pranav/Repositories/zcm/include/ json.hpp	133
/home/pranav/Repositories/zcm/include/ operation_queue.hpp	138
This file declares the Operation_Queue class	
/home/pranav/Repositories/zcm/include/ operation_types.hpp	139
This file declares Operation Types	
/home/pranav/Repositories/zcm/include/ publisher.hpp	140
This file declares the Publisher class	
/home/pranav/Repositories/zcm/include/ server.hpp	140
This file declares the Server class	
/home/pranav/Repositories/zcm/include/ subscriber.hpp	141
This file declares the Subscriber class	
/home/pranav/Repositories/zcm/include/ timer.hpp	142
This file declares the Timer class	
/home/pranav/Repositories/zcm/src/ actor.cpp	143
This file contains definitions for the Actor class	
/home/pranav/Repositories/zcm/src/ client.cpp	143
This file contains definitions for the Client class	
/home/pranav/Repositories/zcm/src/ component.cpp	144
This file contains definitions for the Component class	
/home/pranav/Repositories/zcm/src/ json.cpp	144
/home/pranav/Repositories/zcm/src/ operation_queue.cpp	147
This file contains definitions for the Operation_Queue class	
/home/pranav/Repositories/zcm/src/ operation_types.cpp	147
This file contains definitions for various Operation Types	
/home/pranav/Repositories/zcm/src/ publisher.cpp	148
This file contains definitions for the Publisher class	
/home/pranav/Repositories/zcm/src/ server.cpp	148
This file contains definitions for the Server class	
/home/pranav/Repositories/zcm/src/ subscriber.cpp	149
This file contains definitions for the Subscriber class	
/home/pranav/Repositories/zcm/src/ timer.cpp	149
This file contains definitions for the Timer class	

Chapter 6

Namespace Documentation

6.1 Json Namespace Reference

JSON (JavaScript Object Notation).

Classes

- struct [BuiltStyledStreamWriter](#)
- class [CharReader](#)
Interface for reading JSON from a char array.
- class [CharReaderBuilder](#)
Build a [CharReader](#) implementation.
- struct [CommentStyle](#)
Scoped enums are not available until C++11.
- class [Exception](#)
Base class for all exceptions we throw.
- class [FastWriter](#)
Outputs a [Value](#) in [JSON](#) format without formatting (not human friendly).
- class [Features](#)
Configuration passed to reader and writer.
- class [LogicError](#)
Exceptions thrown by JSON_ASSERT/JSON_FAIL macros.
- class [OurCharReader](#)
- class [OurFeatures](#)
- class [OurReader](#)
- class [Path](#)
Experimental and untested: represents a "path" to access a node.
- class [PathArgument](#)
Experimental and untested: represents an element of the "path" to access a node.
- class [Reader](#)
Unserialize a [JSON](#) document into a [Value](#).
- class [RuntimeError](#)
Exceptions which the user cannot easily avoid.
- class [StaticString](#)
Lightweight wrapper to tag static string.

- class [StreamWriter](#)
Usage:
- class [StreamWriterBuilder](#)
Build a [StreamWriter](#) implementation.
- class [StyledStreamWriter](#)
Writes a [Value](#) in [JSON](#) format in a human friendly way, to a stream rather than to a string.
- class [StyledWriter](#)
Writes a [Value](#) in [JSON](#) format in a human friendly way.
- class [Value](#)
Represents a [JSON](#) value.
- class [ValueConstIterator](#)
const iterator for object and array value.
- class [ValueIterator](#)
Iterator for object and array value.
- class [ValueIteratorBase](#)
base class for [Value](#) iterators.
- class [Writer](#)
Abstract class for writers.

Typedefs

- typedef char [UIntToStringBuffer](#)[[uintToStringBufferSize](#)]
- typedef std::auto_ptr< [CharReader](#) > [CharReaderPtr](#)
- typedef std::auto_ptr< [StreamWriter](#) > [StreamWriterPtr](#)
- typedef int [Int](#)
- typedef unsigned int [UInt](#)
- typedef long long int [Int64](#)
- typedef unsigned long long int [UInt64](#)
- typedef [Int64](#) [LargestInt](#)
- typedef [UInt64](#) [LargestUInt](#)
- typedef unsigned int [ArrayIndex](#)

Enumerations

- enum { [uintToStringBufferSize](#) = 3 * sizeof(LargestUInt) + 1 }
- enum [ValueType](#) {
 [nullValue](#) = 0, [intValue](#), [uintValue](#), [realValue](#),
 [stringValue](#), [booleanValue](#), [arrayValue](#), [objectValue](#) }
- enum [CommentPlacement](#) { [commentBefore](#) = 0, [commentAfterOnSameLine](#), [commentAfter](#), [numberOfComments](#) }

Functions

- static [JSONCPP_STRING](#) [codePointToUTF8](#) (unsigned int cp)
Converts a unicode code-point to UTF-8.
- static bool [isControlCharacter](#) (char ch)
Returns true if ch is a control character (in range [1,31]).
- static void [uintToString](#) ([LargestUInt](#) value, char *¤t)
Converts an unsigned integer to string.
- static void [fixNumericLocale](#) (char *begin, char *end)
Change ',' to '.
- static bool [containsNewLine](#) ([Reader::Location](#) begin, [Reader::Location](#) end)
- static [JSONCPP_STRING](#) [normalizeEOL](#) ([Reader::Location](#) begin, [Reader::Location](#) end)
- static void [getValidReaderKeys](#) (std::set< [JSONCPP_STRING](#) > *valid_keys)
- bool [parseFromStream](#) ([CharReader::Factory](#) const &fact, [JSONCPP_ISTREAM](#) &sin, [Value](#) *root, [JSONCPP_STRING](#) *errs)
- [JSONCPP_ISTREAM](#) & [operator>>](#) ([JSONCPP_ISTREAM](#) &, [Value](#) &)
Read from 'sin' into 'root'.
- static const unsigned char [ALIGNAS](#) (8) kNull[sizeof([Value](#))]
- template<typename T, typename U >
static bool [InRange](#) (double d, T min, U max)
- static char * [duplicateStringValue](#) (const char *value, size_t length)
Duplicates the specified string value.
- static char * [duplicateAndPrefixStringValue](#) (const char *value, unsigned int length)
- static void [decodePrefixedString](#) (bool isPrefixed, char const *prefixed, unsigned *length, char const **value)
- static void [releasePrefixedStringValue](#) (char *value)
Free the string duplicated by [duplicateStringValue\(\)](#)/[duplicateAndPrefixStringValue\(\)](#).
- static void [releaseStringValue](#) (char *value, unsigned)
- [JSONCPP_NORETURN](#) void [throwRuntimeError](#) ([JSONCPP_STRING](#) const &msg)
used internally
- [JSONCPP_NORETURN](#) void [throwLogicError](#) ([JSONCPP_STRING](#) const &msg)
used internally
- static bool [IsIntegral](#) (double d)
- static bool [containsControlCharacter](#) (const char *str)
- static bool [containsControlCharacter0](#) (const char *str, unsigned len)
- [JSONCPP_STRING](#) [valueToString](#) ([LargestInt](#) value)
- [JSONCPP_STRING](#) [valueToString](#) ([LargestUInt](#) value)
- [JSONCPP_STRING](#) [valueToString](#) (double value, bool useSpecialFloats, unsigned int precision)
- [JSONCPP_STRING](#) [valueToString](#) (double value)
- [JSONCPP_STRING](#) [valueToString](#) (bool value)
- [JSONCPP_STRING](#) [valueToQuotedString](#) (const char *value)
- static char const * [strnpbrk](#) (char const *s, char const *accept, size_t n)
- static [JSONCPP_STRING](#) [valueToQuotedStringN](#) (const char *value, unsigned length)
- static void [getValidWriterKeys](#) (std::set< [JSONCPP_STRING](#) > *valid_keys)
- [JSONCPP_STRING](#) [writeString](#) ([StreamWriter::Factory](#) const &factory, [Value](#) const &root)
Write into stringstream, then return string, for convenience.
- [JSONCPP_OSTREAM](#) & [operator<<](#) ([JSONCPP_OSTREAM](#) &, const [Value](#) &root)
Output using the [StyledStreamWriter](#).
- bool [JSON_API](#) [parseFromStream](#) ([CharReader::Factory](#) const &, [JSONCPP_ISTREAM](#) &, [Value](#) *root, std::string *errs)
Consume entire stream and use its begin/end.
- [JSONCPP_STRING](#) [JSON_API](#) [valueToString](#) ([Int](#) value)
- [JSONCPP_STRING](#) [JSON_API](#) [valueToString](#) ([UInt](#) value)

Variables

- const unsigned char & [kNullRef](#) = kNull[0]

6.1.1 Detailed Description

JSON (JavaScript Object Notation).

6.1.2 Typedef Documentation

6.1.2.1 typedef unsigned int [Json::ArrayIndex](#)

6.1.2.2 typedef std::auto_ptr<[CharReader](#)> [Json::CharReaderPtr](#)

6.1.2.3 typedef int [Json::Int](#)

6.1.2.4 typedef long long int [Json::Int64](#)

6.1.2.5 typedef [Int64](#) [Json::LargestInt](#)

6.1.2.6 typedef [UInt64](#) [Json::LargestUInt](#)

6.1.2.7 typedef std::auto_ptr<[StreamWriter](#)> [Json::StreamWriterPtr](#)

6.1.2.8 typedef unsigned int [Json::UInt](#)

6.1.2.9 typedef unsigned long long int [Json::UInt64](#)

6.1.2.10 typedef char [Json::UIntToStringBuffer](#)[[uintToStringBufferSize](#)]

6.1.3 Enumeration Type Documentation

6.1.3.1 anonymous enum

Enumerator

uintToStringBufferSize Constant that specify the size of the buffer that must be passed to [uintToString](#).

6.1.3.2 enum [Json::CommentPlacement](#)

Enumerator

commentBefore a comment placed on the line before a value

commentAfterOnSameLine a comment just after a value on the same line

commentAfter a comment on the line after a value (only make sense for

numberOfCommentPlacement root value)

6.1.3.3 enum `Json::ValueType`

Type of the value held by a [Value](#) object.

Enumerator

nullValue 'null' value
intValue signed integer value
uintValue unsigned integer value
realValue double value
stringValue UTF-8 string value.
booleanValue bool value
arrayValue array value (ordered list)
objectValue object value (collection of name/value pairs).

6.1.4 Function Documentation

6.1.4.1 `static const unsigned char Json::ALIGNAS (8) [static]`

6.1.4.2 `static JSONCPP_STRING Json::codePointToUTF8 (unsigned int cp) [inline],[static]`

Converts a unicode code-point to UTF-8.

6.1.4.3 `static bool Json::containsControlCharacter (const char * str) [static]`

6.1.4.4 `static bool Json::containsControlCharacter0 (const char * str, unsigned len) [static]`

6.1.4.5 `static bool Json::containsNewLine (Reader::Location begin, Reader::Location end) [static]`

6.1.4.6 `static void Json::decodePrefixedString (bool isPrefixed, char const * prefixed, unsigned * length, char const ** value) [inline],[static]`

6.1.4.7 `static char* Json::duplicateAndPrefixStringValue (const char * value, unsigned int length) [inline],[static]`

6.1.4.8 `static char* Json::duplicateStringValue (const char * value, size_t length) [inline],[static]`

Duplicates the specified string value.

Parameters

<i>value</i>	Pointer to the string to duplicate. Must be zero-terminated if length is "unknown".
<i>length</i>	Length of the value. if equals to unknown, then it will be computed using strlen(value).

Returns

Pointer on the duplicate instance of string.

6.1.4.9 `static void Json::fixNumericLocale (char * begin, char * end)` `[inline], [static]`

Change ',' to '.

' everywhere in buffer.

We had a sophisticated way, but it did not work in WinCE.

See also

<https://github.com/open-source-parsers/jsoncpp/pull/9>

6.1.4.10 `static void Json::getValidReaderKeys (std::set< JSONCPP_STRING > * valid_keys)` `[static]`

6.1.4.11 `static void Json::getValidWriterKeys (std::set< JSONCPP_STRING > * valid_keys)` `[static]`

6.1.4.12 `template<typename T, typename U > static bool Json::InRange (double d, T min, U max)` `[inline], [static]`

6.1.4.13 `static bool Json::isControlCharacter (char ch)` `[inline], [static]`

Returns true if *ch* is a control character (in range [1,31]).

6.1.4.14 `static bool Json::isIntegral (double d)` `[static]`

6.1.4.15 `static JSONCPP_STRING Json::normalizeEOL (Reader::Location begin, Reader::Location end)` `[static]`

6.1.4.16 `JSON_API JSONCPP_OSTREAM & Json::operator<< (JSONCPP_OSTREAM &, const Value & root)`

Output using the [StyledStreamWriter](#).

See also

[Json::operator>>\(\)](#)

6.1.4.17 `JSON_API JSONCPP_ISTREAM & Json::operator>> (JSONCPP_ISTREAM &, Value &)`

Read from 'sin' into 'root'.

Always keep comments from the input JSON.

This can be used to read a file into a particular sub-object. For example:

```
Json::Value root;
cin >> root["dir"]["file"];
cout << root;
```

Result:

```
{
  "dir": {
    "file": {
      // The input stream JSON would be nested here.
    }
  }
}
```

Exceptions

<code>std::exception</code>	on parse error.
-----------------------------	-----------------

See also

[Json::operator<<\(\)](#)

6.1.4.18 `bool JSON_API Json::parseFromStream (CharReader::Factory const &, JSONCPP_ISTREAM &, Value * root, std::string * errs)`

Consume entire stream and use its begin/end.

Someday we might have a real StreamReader, but for now this is convenient.

6.1.4.19 `bool Json::parseFromStream (CharReader::Factory const & fact, JSONCPP_ISTREAM & sin, Value * root, JSONCPP_STRING * errs)`

6.1.4.20 `static void Json::releasePrefixedStringValue (char * value)` `[inline],[static]`

Free the string duplicated by [duplicateStringValue\(\)](#)/[duplicateAndPrefixStringValue\(\)](#).

6.1.4.21 `static void Json::releaseStringValue (char * value, unsigned)` `[inline],[static]`

6.1.4.22 `static char const* Json::strnpbrk (char const * s, char const * accept, size_t n)` `[static]`

6.1.4.23 `JSONCPP_NORETURN void Json::throwLogicError (JSONCPP_STRING const & msg)`

used internally

6.1.4.24 `JSONCPP_NORETURN void Json::throwRuntimeError (JSONCPP_STRING const & msg)`

used internally

6.1.4.25 `static void Json::uintToString (LargestUInt value, char *& current)` `[inline],[static]`

Converts an unsigned integer to string.

Parameters

<i>value</i>	Unsigned interger to convert to string
<i>current</i>	Input/Output string buffer. Must have at least <code>uintToStringBufferSize</code> chars free.

- 6.1.4.26 JSONCPP_STRING JSON_API Json::valueToQuotedString (const char * *value*)
- 6.1.4.27 static JSONCPP_STRING Json::valueToQuotedStringN (const char * *value*, unsigned *length*) [static]
- 6.1.4.28 JSONCPP_STRING JSON_API Json::valueToString (Int *value*)
- 6.1.4.29 JSONCPP_STRING JSON_API Json::valueToString (UInt *value*)
- 6.1.4.30 JSONCPP_STRING JSON_API Json::valueToString (LargestInt *value*)
- 6.1.4.31 JSONCPP_STRING JSON_API Json::valueToString (LargestUInt *value*)
- 6.1.4.32 JSONCPP_STRING Json::valueToString (double *value*, bool *useSpecialFloats*, unsigned int *precision*)
- 6.1.4.33 JSONCPP_STRING JSON_API Json::valueToString (double *value*)
- 6.1.4.34 JSONCPP_STRING JSON_API Json::valueToString (bool *value*)
- 6.1.4.35 JSONCPP_STRING JSON_API Json::writeString (StreamWriter::Factory const & *factory*, Value const & *root*)

Write into stringstream, then return string, for convenience.

A [StreamWriter](#) will be created from the factory, used, and then deleted.

6.1.5 Variable Documentation

- 6.1.5.1 const unsigned char& Json::kNullRef = kNull[0]

6.2 std Namespace Reference

Functions

- template<>
void [swap](#) (Json::Value &a, Json::Value &b)
Specialize [std::swap\(\)](#) for [Json::Value](#).

6.2.1 Function Documentation

- 6.2.1.1 template<> void std::swap (Json::Value & *a*, Json::Value & *b*) [inline]

Specialize [std::swap\(\)](#) for [Json::Value](#).

6.3 zcm Namespace Reference

Classes

- class [Actor](#)
Actor class.
- class [Base_Operation](#)
Base Operation class.
- class [Client](#)
Client class.
- class [Component](#)
Component class.
- class [Operation_Queue](#)
Operation_Queue class.
- class [Publisher](#)
Publisher class.
- class [Server](#)
Server class.
- class [Server_Operation](#)
Server Operation class.
- class [Subscriber](#)
Subscriber class.
- class [Subscriber_Operation](#)
Subscriber Operation class.
- class [Timer](#)
Timer class.
- class [Timer_Operation](#)
Timer Operation class.

Chapter 7

Class Documentation

7.1 zcm::Actor Class Reference

[Actor](#) class.

```
#include <actor.hpp>
```

Public Member Functions

- void [configure](#) (std::string configuration_file)
Configure the component_instances vector.
- void [run](#) ()
Spawn all component instances.
- std::string [get_name](#) ()
Get actor name.

Private Attributes

- std::string [name](#)
- std::vector< [Component](#) * > [component_instances](#)

7.1.1 Detailed Description

[Actor](#) class.

7.1.2 Member Function Documentation

7.1.2.1 void zcm::Actor::configure (std::string configuration_file)

Configure the component_instances vector.

Parameters

in	<i>configuration_file</i>	JSON configuration file to parse
----	---------------------------	----------------------------------

7.1.2.2 `std::string zcm::Actor::get_name ()`

Get actor name.

Returns

Name of the actor

7.1.2.3 `void zcm::Actor::run ()`

Spawn all component instances.

7.1.3 Member Data Documentation

7.1.3.1 `std::vector<Component*> zcm::Actor::component_instances` [private]7.1.3.2 `std::string zcm::Actor::name` [private]

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

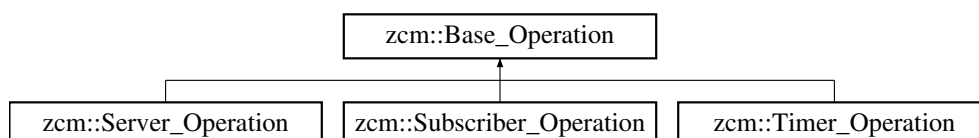
- [/home/pranav/Repositories/zcm/include/actor.hpp](#)
- [/home/pranav/Repositories/zcm/src/actor.cpp](#)

7.2 `zcm::Base_Operation` Class Reference

Base Operation class.

```
#include <operation_types.hpp>
```

Inheritance diagram for `zcm::Base_Operation`:



Public Member Functions

- [Base_Operation](#) (std::string [name](#), unsigned int [priority](#))
Construct a base operation.
- std::string [get_name](#) ()
Return the operation name.
- unsigned int [get_priority](#) () const
Return the operation priority.
- virtual void [execute](#) ()
Virtual execute function overridden by concrete types.

Private Attributes

- std::string [name](#)
Name of the Operation.
- unsigned int [priority](#)
Priority of the Operation.

7.2.1 Detailed Description

Base Operation class.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 `zcm::Base_Operation::Base_Operation (std::string name, unsigned int priority)` `[inline]`

Construct a base operation.

Parameters

in	<i>name</i>	Name of the operation
in	<i>priority</i>	Priority of the operation

7.2.3 Member Function Documentation

7.2.3.1 `virtual void zcm::Base_Operation::execute ()` `[inline]`, `[virtual]`

Virtual execute function overridden by concrete types.

Reimplemented in [zcm::Server_Operation](#), [zcm::Subscriber_Operation](#), and [zcm::Timer_Operation](#).

7.2.3.2 `std::string zcm::Base_Operation::get_name ()`

Return the operation name.

Returns

Name of the operation

7.2.3.3 unsigned int zcm::Base_Operation::get_priority () const

Return the operation priority.

Returns

Priority of the operation

7.2.4 Member Data Documentation

7.2.4.1 std::string zcm::Base_Operation::name [private]

Name of the Operation.

7.2.4.2 unsigned int zcm::Base_Operation::priority [private]

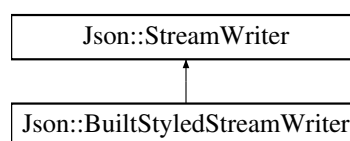
Priority of the Operation.

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

- [/home/pranav/Repositories/zcm/include/operation_types.hpp](#)
- [/home/pranav/Repositories/zcm/src/operation_types.cpp](#)

7.3 Json::BuiltStyledStreamWriter Struct Reference

Inheritance diagram for Json::BuiltStyledStreamWriter:



Public Member Functions

- [BuiltStyledStreamWriter](#) ([JSONCPP_STRING](#) const &indentation, [CommentStyle::Enum](#) cs, [JSONCPP_STRING](#) const &colonSymbol, [JSONCPP_STRING](#) const &>nullSymbol, [JSONCPP_STRING](#) const &endingLineFeedSymbol, bool useSpecialFloats, unsigned int precision)
- int [write](#) ([Value](#) const &root, [JSONCPP_OSTREAM](#) *sout) [JSONCPP_OVERRIDE](#)
Write [Value](#) into document as configured in sub-class.

Protected Attributes

- [JSONCPP_OSTREAM](#) * sout_

Private Types

- typedef std::vector< JSONCPP_STRING > ChildValues

Private Member Functions

- void writeValue (Value const &value)
- void writeArrayValue (Value const &value)
- bool isMultilineArray (Value const &value)
- void pushValue (JSONCPP_STRING const &value)
- void writeIndent ()
- void writeWithIndent (JSONCPP_STRING const &value)
- void indent ()
- void unindent ()
- void writeCommentBeforeValue (Value const &root)
- void writeCommentAfterValueOnSameLine (Value const &root)

Static Private Member Functions

- static bool hasCommentForValue (const Value &value)

Private Attributes

- ChildValues childValues_
- JSONCPP_STRING indentString_
- unsigned int rightMargin_
- JSONCPP_STRING indentation_
- CommentStyle::Enum cs_
- JSONCPP_STRING colonSymbol_
- JSONCPP_STRING nullSymbol_
- JSONCPP_STRING endingLineFeedSymbol_
- bool addChildValues_: 1
- bool indented_: 1
- bool useSpecialFloats_: 1
- unsigned int precision_

7.3.1 Member Typedef Documentation

- 7.3.1.1 typedef std::vector<JSONCPP_STRING> Json::BuiltStyledStreamWriter::ChildValues [private]

7.3.2 Constructor & Destructor Documentation

- 7.3.2.1 Json::BuiltStyledStreamWriter::BuiltStyledStreamWriter (JSONCPP_STRING const & *indentation*, CommentStyle::Enum *cs*, JSONCPP_STRING const & *colonSymbol*, JSONCPP_STRING const & *nullSymbol*, JSONCPP_STRING const & *endingLineFeedSymbol*, bool *useSpecialFloats*, unsigned int *precision*)

7.3.3 Member Function Documentation

7.3.3.1 `bool Json::BuiltStyledStreamWriter::hasCommentForValue (const Value & value) [static], [private]`

7.3.3.2 `void Json::BuiltStyledStreamWriter::indent () [private]`

7.3.3.3 `bool Json::BuiltStyledStreamWriter::isMultilineArray (Value const & value) [private]`

7.3.3.4 `void Json::BuiltStyledStreamWriter::pushValue (JSONCPP_STRING const & value) [private]`

7.3.3.5 `void Json::BuiltStyledStreamWriter::unindent () [private]`

7.3.3.6 `int Json::BuiltStyledStreamWriter::write (Value const & root, JSONCPP_OSTREAM * sout) [virtual]`

Write [Value](#) into document as configured in sub-class.

Do not take ownership of sout, but maintain a reference during function.

Precondition

sout != NULL

Returns

zero on success (For now, we always return zero, so check the stream instead.)

Exceptions

<code>std::exception</code>	possibly, depending on configuration
-----------------------------	--------------------------------------

Implements [Json::StreamWriter](#).

7.3.3.7 `void Json::BuiltStyledStreamWriter::writeArrayValue (Value const & value) [private]`

7.3.3.8 `void Json::BuiltStyledStreamWriter::writeCommentAfterValueOnSameLine (Value const & root) [private]`

7.3.3.9 `void Json::BuiltStyledStreamWriter::writeCommentBeforeValue (Value const & root) [private]`

7.3.3.10 `void Json::BuiltStyledStreamWriter::writeIndent () [private]`

7.3.3.11 `void Json::BuiltStyledStreamWriter::writeValue (Value const & value) [private]`

7.3.3.12 `void Json::BuiltStyledStreamWriter::writeWithIndent (JSONCPP_STRING const & value) [private]`

7.3.4 Member Data Documentation

7.3.4.1 `bool Json::BuiltStyledStreamWriter::addChildValues_ [private]`

- 7.3.4.2 **ChildValues** `Json::BuiltStyledStreamWriter::childValues_` [private]
- 7.3.4.3 **JSONCPP_STRING** `Json::BuiltStyledStreamWriter::colonSymbol_` [private]
- 7.3.4.4 **CommentStyle::Enum** `Json::BuiltStyledStreamWriter::cs_` [private]
- 7.3.4.5 **JSONCPP_STRING** `Json::BuiltStyledStreamWriter::endingLineFeedSymbol_` [private]
- 7.3.4.6 **JSONCPP_STRING** `Json::BuiltStyledStreamWriter::indentation_` [private]
- 7.3.4.7 **bool** `Json::BuiltStyledStreamWriter::indented_` [private]
- 7.3.4.8 **JSONCPP_STRING** `Json::BuiltStyledStreamWriter::indentString_` [private]
- 7.3.4.9 **JSONCPP_STRING** `Json::BuiltStyledStreamWriter::nullSymbol_` [private]
- 7.3.4.10 **unsigned int** `Json::BuiltStyledStreamWriter::precision_` [private]
- 7.3.4.11 **unsigned int** `Json::BuiltStyledStreamWriter::rightMargin_` [private]
- 7.3.4.12 **JSONCPP_OSTREAM*** `Json::StreamWriter::sout_` [protected],[inherited]
- 7.3.4.13 **bool** `Json::BuiltStyledStreamWriter::useSpecialFloats_` [private]

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

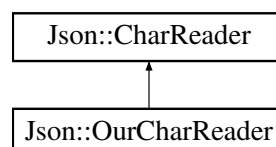
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.4 Json::CharReader Class Reference

Interface for reading JSON from a char array.

```
#include <json.hpp>
```

Inheritance diagram for `Json::CharReader`:



Classes

- class [Factory](#)

Public Member Functions

- virtual [~CharReader](#) ()
- virtual bool [parse](#) (char const *beginDoc, char const *endDoc, [Value](#) *root, [JSONCPP_STRING](#) *errs)=0
Read a [Value](#) from a [JSON](#) document.

7.4.1 Detailed Description

Interface for reading JSON from a char array.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 virtual [Json::CharReader::~~CharReader](#) () [inline], [virtual]

7.4.3 Member Function Documentation

7.4.3.1 virtual bool [Json::CharReader::parse](#) (char const * *beginDoc*, char const * *endDoc*, [Value](#) * *root*, [JSONCPP_STRING](#) * *errs*) [pure virtual]

Read a [Value](#) from a [JSON](#) document.

The document must be a UTF-8 encoded string containing the document to read.

Parameters

<i>beginDoc</i>	Pointer on the beginning of the UTF-8 encoded string of the document to read.
<i>endDoc</i>	Pointer on the end of the UTF-8 encoded string of the document to read. Must be >= beginDoc.
<i>root</i>	[out] Contains the root value of the document if it was successfully parsed.
<i>errs</i>	[out] Formatted error messages (if not NULL) a user friendly string that lists errors in the parsed document.

Returns

`true` if the document was successfully parsed, `false` if an error occurred.

Implemented in [Json::OurCharReader](#).

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

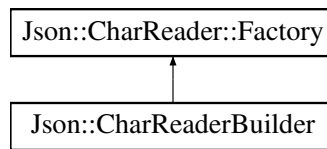
- [/home/pranav/Repositories/zcm/include/json.hpp](#)

7.5 Json::CharReaderBuilder Class Reference

Build a [CharReader](#) implementation.

```
#include <json.hpp>
```

Inheritance diagram for Json::CharReaderBuilder:



Public Member Functions

- [CharReaderBuilder](#) ()
- [~CharReaderBuilder](#) () `JSONCPP_OVERRIDE`
- [CharReader](#) * [newCharReader](#) () const `JSONCPP_OVERRIDE`
Allocate a [CharReader](#) via operator new().
- bool [validate](#) (Json::Value *invalid) const
- [Value](#) & [operator\[\]](#) (JSONCPP_STRING key)
A simple way to update a specific setting.

Static Public Member Functions

- static void [setDefaults](#) (Json::Value *settings)
Called by ctor, but you can use this to reset settings_.
- static void [strictMode](#) (Json::Value *settings)
Same as old [Features::strictMode\(\)](#).

Public Attributes

- [Json::Value](#) [settings_](#)
Configuration of this builder.

7.5.1 Detailed Description

Build a [CharReader](#) implementation.

Usage:

```
using namespace Json;
CharReaderBuilder builder;
builder["collectComments"] = false;
Value value;
JSONCPP_STRING errs;
bool ok = parseFromStream(builder, std::cin, &value, &errs);
```

7.5.2 Constructor & Destructor Documentation

7.5.2.1 [Json::CharReaderBuilder::CharReaderBuilder](#) ()

7.5.2.2 [Json::CharReaderBuilder::~~CharReaderBuilder](#) ()

7.5.3 Member Function Documentation

7.5.3.1 [CharReader](#) * [Json::CharReaderBuilder::newCharReader](#) () const `[virtual]`

Allocate a [CharReader](#) via operator new().

Exceptions

<code>std::exception</code>	if something goes wrong (e.g. invalid settings)
-----------------------------	---

Implements [Json::CharReader::Factory](#).

7.5.3.2 Value & Json::CharReaderBuilder::operator[] (JSONCPP_STRING key)

A simple way to update a specific setting.

7.5.3.3 void Json::CharReaderBuilder::setDefaults (Json::Value * settings) [static]

Called by ctor, but you can use this to reset settings_.

Precondition

'settings' != NULL (but Json::null is fine)

Remarks

Defaults:

[CharReaderBuilderDefaults]

[CharReaderBuilderDefaults]

7.5.3.4 void Json::CharReaderBuilder::strictMode (Json::Value * settings) [static]

Same as old [Features::strictMode\(\)](#).

Precondition

'settings' != NULL (but Json::null is fine)

Remarks

Defaults:

[CharReaderBuilderStrictMode]

[CharReaderBuilderStrictMode]

7.5.3.5 `bool Json::CharReaderBuilder::validate (Json::Value * invalid) const`

Returns

true if 'settings' are legal and consistent; otherwise, indicate bad settings via 'invalid'.

7.5.4 Member Data Documentation

7.5.4.1 `Json::Value Json::CharReaderBuilder::settings_`

Configuration of this builder.

These are case-sensitive. Available settings (case-sensitive):

- `"collectComments": false or true`
 - true to collect comment and allow writing them back during serialization, false to discard comments. This parameter is ignored if allowComments is false.
- `"allowComments": false or true`
 - true if comments are allowed.
- `"strictRoot": false or true`
 - true if root must be either an array or an object value
- `"allowDroppedNullPlaceholders": false or true`
 - true if dropped null placeholders are allowed. (See [StreamWriterBuilder](#).)
- `"allowNumericKeys": false or true`
 - true if numeric object keys are allowed.
- `"allowSingleQuotes": false or true`
 - true if " are allowed for strings (both keys and values)
- `"stackLimit": integer`
 - Exceeding stackLimit (recursive depth of `readValue()`) will cause an exception.
 - This is a security issue (seg-faults caused by deeply nested JSON), so the default is low.
- `"failIfExtra": false or true`
 - If true, `parse()` returns false when extra non-whitespace trails the JSON value in the input string.
- `"rejectDupKeys": false or true`
 - If true, `parse()` returns false when a key is duplicated within an object.
- `"allowSpecialFloats": false or true`
 - If true, special float values (NaNs and infinities) are allowed and their values are lossfree restorable.

You can examine 'settings_' yourself to see the defaults. You can also write and read them just like any JSON [Value](#).

See also

[setDefaults\(\)](#)

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.6 zcm::Client Class Reference

[Client](#) class.

```
#include <client.hpp>
```

Public Member Functions

- [Client](#) (std::string [name](#))
Construct a client object.
- [Client](#) (std::string [name](#), std::vector< std::string > [endpoints](#))
Construct a client object with known endpoints.
- [~Client](#) ()
Close the client ZMQ socket and destroy the context.
- void [connect](#) (std::vector< std::string > [new_endpoints](#))
Connect the client to a new set of endpoints.
- std::string [get_name](#) ()
Return the client name.
- std::string [call](#) (std::string message)
Call the server.

Private Attributes

- std::string [name](#)
Name of the publisher.
- std::vector< std::string > [endpoints](#)
Vector of endpoints to connect to.
- zmq::context_t * [context](#)
ZMQ Context of the client.
- zmq::socket_t * [client_socket](#)
ZMQ Socket of the client.

7.6.1 Detailed Description

[Client](#) class.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 zcm::Client::Client (std::string *name*)

Construct a client object.

Parameters

in	<i>name</i>	Client name
----	-------------	-----------------------------

7.6.2.2 zcm::Client::Client (std::string *name*, std::vector< std::string > *endpoints*)

Construct a client object with known endpoints.

Parameters

in	<i>name</i>	Client name
in	<i>endpoints</i>	A vector of endpoint strings

7.6.2.3 zcm::Client::~~Client ()

Close the client ZMQ socket and destroy the context.

7.6.3 Member Function Documentation

7.6.3.1 std::string zcm::Client::call (std::string *message*)

Call the server.

Parameters

in	<i>message</i>	The message string. Serialize complex objects to strings with protobuf
----	----------------	--

7.6.3.2 void zcm::Client::connect (std::vector< std::string > *new_endpoints*)

Connect the client to a new set of endpoints.

Parameters

in	<i>new_endpoints</i>	New set of endpoints as a vector
----	----------------------	----------------------------------

7.6.3.3 std::string zcm::Client::get_name ()

Return the client name.

Returns

Client name

7.6.4 Member Data Documentation

7.6.4.1 zmq::socket_t* zcm::Client::client_socket [private]

ZMQ Socket of the client.

7.6.4.2 `zmq::context_t* zcm::Client::context` [private]

ZMQ Context of the client.

7.6.4.3 `std::vector<std::string> zcm::Client::endpoints` [private]

Vector of endpoints to connect to.

7.6.4.4 `std::string zcm::Client::name` [private]

Name of the publisher.

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

- [/home/pranav/Repositories/zcm/include/client.hpp](#)
- [/home/pranav/Repositories/zcm/src/client.cpp](#)

7.7 `Json::Value::CommentInfo` Struct Reference

Public Member Functions

- [CommentInfo](#) ()
- [~CommentInfo](#) ()
- void [setComment](#) (const char *text, size_t len)

Public Attributes

- char * [comment_](#)

7.7.1 Constructor & Destructor Documentation

7.7.1.1 `Json::Value::CommentInfo::CommentInfo ()`

7.7.1.2 `Json::Value::CommentInfo::~~CommentInfo ()`

7.7.2 Member Function Documentation

7.7.2.1 `void Json::Value::CommentInfo::setComment (const char * text, size_t len)`

7.7.3 Member Data Documentation

7.7.3.1 `char* Json::Value::CommentInfo::comment_`

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.8 Json::CommentStyle Struct Reference

Scoped enums are not available until C++11.

Public Types

- enum [Enum](#) { [None](#), [Most](#), [All](#) }
Decide whether to write comments.

7.8.1 Detailed Description

Scoped enums are not available until C++11.

7.8.2 Member Enumeration Documentation

7.8.2.1 enum Json::CommentStyle::Enum

Decide whether to write comments.

Enumerator

- None** Drop all comments.
- Most** Recover odd behavior of previous versions (not implemented yet).
- All** Keep all comments.

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

- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.9 zcm::Component Class Reference

[Component](#) class.

```
#include <component.hpp>
```

Public Member Functions

- [Component](#) ()
Construct a component Prepare the component operation queue.
- [~Component](#) ()
Destroy the component.
- std::thread * [spawn](#) ()
Spawn the component executor thread.

Protected Attributes

- [Operation_Queue](#) * [operation_queue](#)
Pointer to the [Component](#) Operation Queue.
- `std::thread` * [executor_thread](#)
Pointer to the [Component](#) Executor Thread.

7.9.1 Detailed Description

[Component](#) class.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 `zcm::Component::Component ()`

Construct a component Prepare the component operation queue.

7.9.2.2 `zcm::Component::~~Component ()`

Destroy the component.

7.9.3 Member Function Documentation

7.9.3.1 `std::thread* zcm::Component::spawn ()`

Spawn the component executor thread.

Returns

Return a pointer to the executor thread

7.9.4 Member Data Documentation

7.9.4.1 `std::thread* zcm::Component::executor_thread` [protected]

Pointer to the [Component](#) Executor Thread.

7.9.4.2 `Operation_Queue* zcm::Component::operation_queue` [protected]

Pointer to the [Component](#) Operation Queue.

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

- `/home/pranav/Repositories/zcm/include/component.hpp`
- `/home/pranav/Repositories/zcm/src/component.cpp`

7.10 Json::Value::CZString Class Reference

Classes

- struct [StringStorage](#)

Public Types

- enum [DuplicationPolicy](#) { [noDuplication](#) = 0, [duplicate](#), [duplicateOnCopy](#) }

Public Member Functions

- [CZString](#) ([ArrayIndex](#) index)
- [CZString](#) (char const *str, unsigned [length](#), [DuplicationPolicy](#) allocate)
- [CZString](#) ([CZString](#) const &other)
- [~CZString](#) ()
- [CZString](#) & [operator=](#) ([CZString](#) other)
- bool [operator<](#) ([CZString](#) const &other) const
- bool [operator==](#) ([CZString](#) const &other) const
- [ArrayIndex](#) [index](#) () const
- char const * [data](#) () const
- unsigned [length](#) () const
- bool [isStaticString](#) () const

Private Member Functions

- void [swap](#) ([CZString](#) &other)

Private Attributes

- char const * [cstr_](#)
- union {
 [ArrayIndex](#) [index_](#)
 [StringStorage](#) [storage_](#)
};

7.10.1 Member Enumeration Documentation

7.10.1.1 enum Json::Value::CZString::DuplicationPolicy

Enumerator

noDuplication
duplicate
duplicateOnCopy

7.10.2 Constructor & Destructor Documentation

7.10.2.1 `Json::Value::CZString::CZString (ArrayIndex index)`

7.10.2.2 `Json::Value::CZString::CZString (char const * str, unsigned length, DuplicationPolicy allocate)`

7.10.2.3 `Json::Value::CZString::CZString (CZString const & other)`

7.10.2.4 `Json::Value::CZString::~~CZString ()`

7.10.3 Member Function Documentation

7.10.3.1 `const char * Json::Value::CZString::data () const`

7.10.3.2 `ArrayIndex Json::Value::CZString::index () const`

7.10.3.3 `bool Json::Value::CZString::isStaticString () const`

7.10.3.4 `unsigned Json::Value::CZString::length () const`

7.10.3.5 `bool Json::Value::CZString::operator< (CZString const & other) const`

7.10.3.6 `Value::CZString & Json::Value::CZString::operator= (CZString other)`

7.10.3.7 `bool Json::Value::CZString::operator== (CZString const & other) const`

7.10.3.8 `void Json::Value::CZString::swap (CZString & other) [private]`

7.10.4 Member Data Documentation

7.10.4.1 `union { ... } [private]`

7.10.4.2 `char const* Json::Value::CZString::cstr_ [private]`

7.10.4.3 `ArrayIndex Json::Value::CZString::index_`

7.10.4.4 `StringStorage Json::Value::CZString::storage_`

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.11 Json::OurReader::ErrorInfo Class Reference

Public Attributes

- [Token token_](#)
- [JSONCPP_STRING message_](#)
- [Location extra_](#)

7.11.1 Member Data Documentation

7.11.1.1 [Location](#) [Json::OurReader::ErrorInfo::extra_](#)

7.11.1.2 [JSONCPP_STRING](#) [Json::OurReader::ErrorInfo::message_](#)

7.11.1.3 [Token](#) [Json::OurReader::ErrorInfo::token_](#)

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

- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.12 Json::Reader::ErrorInfo Class Reference

Public Attributes

- [Token token_](#)
- [JSONCPP_STRING message_](#)
- [Location extra_](#)

7.12.1 Member Data Documentation

7.12.1.1 [Location](#) [Json::Reader::ErrorInfo::extra_](#)

7.12.1.2 [JSONCPP_STRING](#) [Json::Reader::ErrorInfo::message_](#)

7.12.1.3 [Token](#) [Json::Reader::ErrorInfo::token_](#)

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

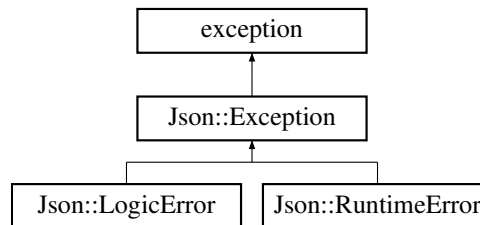
- [/home/pranav/Repositories/zcm/include/json.hpp](#)

7.13 Json::Exception Class Reference

Base class for all exceptions we throw.

```
#include <json.hpp>
```

Inheritance diagram for Json::Exception:



Public Member Functions

- [Exception](#) ([JSONCPP_STRING](#) const &msg)
- [~Exception](#) () [JSONCPP_OVERRIDE](#) throw ()
- char const * [what](#) () const [JSONCPP_OVERRIDE](#) throw ()

Protected Attributes

- [JSONCPP_STRING](#) msg_

7.13.1 Detailed Description

Base class for all exceptions we throw.

We use nothing but these internally. Of course, STL can throw others.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 `Json::Exception::Exception (JSONCPP_STRING const & msg)`

7.13.2.2 `Json::Exception::~~Exception () throw ()`

7.13.3 Member Function Documentation

7.13.3.1 `char const * Json::Exception::what () const throw ()`

7.13.4 Member Data Documentation

7.13.4.1 `JSONCPP_STRING Json::Exception::msg_ [protected]`

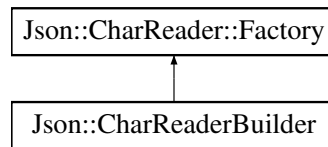
The documentation for this class was generated from the following files:

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.14 Json::CharReader::Factory Class Reference

```
#include <json.hpp>
```

Inheritance diagram for Json::CharReader::Factory:



Public Member Functions

- virtual `~Factory()`
- virtual `CharReader* newCharReader()` const =0
Allocate a `CharReader` via operator `new()`.

7.14.1 Constructor & Destructor Documentation

7.14.1.1 virtual `Json::CharReader::Factory::~~Factory()` [inline],[virtual]

7.14.2 Member Function Documentation

7.14.2.1 virtual `CharReader* Json::CharReader::Factory::newCharReader()` const [pure virtual]

Allocate a `CharReader` via operator `new()`.

Exceptions

<code>std::exception</code>	if something goes wrong (e.g. invalid settings)
-----------------------------	---

Implemented in `Json::CharReaderBuilder`.

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

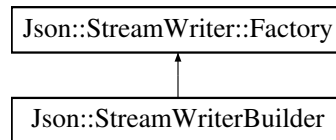
- `/home/pranav/Repositories/zcm/include/json.hpp`

7.15 Json::StreamWriter::Factory Class Reference

A simple abstract factory.

```
#include <json.hpp>
```

Inheritance diagram for Json::StreamWriter::Factory:



Public Member Functions

- virtual `~Factory()`
- virtual `StreamWriter* newStreamWriter() const =0`
Allocate a `CharReader` via operator `new()`.

7.15.1 Detailed Description

A simple abstract factory.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 `Json::StreamWriter::Factory::~~Factory()` [virtual]

7.15.3 Member Function Documentation

7.15.3.1 `virtual StreamWriter* Json::StreamWriter::Factory::newStreamWriter() const` [pure virtual]

Allocate a `CharReader` via operator `new()`.

Exceptions

<code>std::exception</code>	if something goes wrong (e.g. invalid settings)
-----------------------------	---

Implemented in `Json::StreamWriterBuilder`.

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

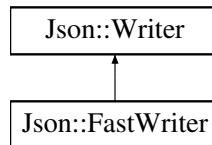
- `/home/pranav/Repositories/zcm/include/json.hpp`
- `/home/pranav/Repositories/zcm/src/json.cpp`

7.16 Json::FastWriter Class Reference

Outputs a `Value` in `JSON` format without formatting (not human friendly).

```
#include <json.hpp>
```

Inheritance diagram for `Json::FastWriter`:



Public Member Functions

- [FastWriter](#) ()
- [~FastWriter](#) () [JSONCPP_OVERRIDE](#)
- void [enableYAMLCompatibility](#) ()
- void [dropNullPlaceholders](#) ()
Drop the "null" string from the writer's output for nullValues.
- void [omitEndingLineFeed](#) ()
- [JSONCPP_STRING](#) [write](#) (const [Value](#) &root) [JSONCPP_OVERRIDE](#)

Private Member Functions

- void [writeValue](#) (const [Value](#) &value)

Private Attributes

- [JSONCPP_STRING](#) [document_](#)
- bool [yamlCompatibilityEnabled_](#)
- bool [dropNullPlaceholders_](#)
- bool [omitEndingLineFeed_](#)

7.16.1 Detailed Description

Outputs a [Value](#) in [JSON](#) format without formatting (not human friendly).

The JSON document is written in a single line. It is not intended for 'human' consumption, but may be usefull to support feature such as RPC where bandwith is limited.

See also

[Reader](#), [Value](#)

Deprecated Use [StreamWriterBuilder](#).

7.16.2 Constructor & Destructor Documentation

7.16.2.1 [Json::FastWriter::FastWriter](#) ()

7.16.2.2 [Json::FastWriter::~~FastWriter](#) () [\[inline\]](#)

7.16.3 Member Function Documentation

7.16.3.1 void [Json::FastWriter::dropNullPlaceholders](#) ()

Drop the "null" string from the writer's output for nullValues.

Strictly speaking, this is not valid JSON. But when the output is being fed to a browser's Javascript, it makes for smaller output and the browser can handle the output just fine.

7.16.3.2 `void Json::FastWriter::enableYAMLCompatibility ()`

7.16.3.3 `void Json::FastWriter::omitEndingLineFeed ()`

7.16.3.4 `JSONCPP_STRING Json::FastWriter::write (const Value & root)` [virtual]

Implements [Json::Writer](#).

7.16.3.5 `void Json::FastWriter::writeValue (const Value & value)` [private]

7.16.4 Member Data Documentation

7.16.4.1 `JSONCPP_STRING Json::FastWriter::document_` [private]

7.16.4.2 `bool Json::FastWriter::dropNullPlaceholders_` [private]

7.16.4.3 `bool Json::FastWriter::omitEndingLineFeed_` [private]

7.16.4.4 `bool Json::FastWriter::yamlCompatibilityEnabled_` [private]

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.17 Json::Features Class Reference

Configuration passed to reader and writer.

```
#include <json.hpp>
```

Public Member Functions

- [Features](#) ()
Initialize the configuration like `JsonConfig::allFeatures`;

Static Public Member Functions

- static [Features all](#) ()
A configuration that allows all features and assumes all strings are UTF-8.
- static [Features strictMode](#) ()
A configuration that is strictly compatible with the JSON specification.

Public Attributes

- bool [allowComments_](#)
true if comments are allowed. Default: true.
- bool [strictRoot_](#)
true if root must be either an array or an object value.
- bool [allowDroppedNullPlaceholders_](#)
true if dropped null placeholders are allowed. Default: false.
- bool [allowNumericKeys_](#)
true if numeric object key are allowed. Default: false.

7.17.1 Detailed Description

Configuration passed to reader and writer.

This configuration object can be used to force the [Reader](#) or [Writer](#) to behave in a standard conforming way.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 `Json::Features::Features ()`

Initialize the configuration like `JsonConfig::allFeatures;`.

7.17.3 Member Function Documentation

7.17.3.1 `Features Json::Features::all () [static]`

A configuration that allows all features and assumes all strings are UTF-8.

- C & C++ comments are allowed
- Root object can be any JSON value
- Assumes [Value](#) strings are encoded in UTF-8

7.17.3.2 `Features Json::Features::strictMode () [static]`

A configuration that is strictly compatible with the JSON specification.

- Comments are forbidden.
- Root object must be either an array or an object value.
- Assumes [Value](#) strings are encoded in UTF-8

7.17.4 Member Data Documentation

7.17.4.1 `bool Json::Features::allowComments_`

`true` if comments are allowed. Default: `true`.

7.17.4.2 `bool Json::Features::allowDroppedNullPlaceholders_`

`true` if dropped null placeholders are allowed. Default: `false`.

7.17.4.3 `bool Json::Features::allowNumericKeys_`

`true` if numeric object key are allowed. Default: `false`.

7.17.4.4 `bool Json::Features::strictRoot_`

`true` if root must be either an array or an object value.

Default: `false`.

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

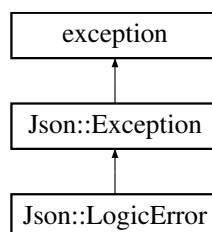
- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.18 `Json::LogicError` Class Reference

Exceptions thrown by `JSON_ASSERT/JSON_FAIL` macros.

```
#include <json.hpp>
```

Inheritance diagram for `Json::LogicError`:



Public Member Functions

- [LogicError](#) ([JSONCPP_STRING](#) const &msg)
- `char const * what () const` [JSONCPP_OVERRIDE](#) `throw ()`

Protected Attributes

- [JSONCPP_STRING msg_](#)

7.18.1 Detailed Description

Exceptions thrown by JSON_ASSERT/JSON_FAIL macros.

These are precondition-violations (user bugs) and internal errors (our bugs).

Remarks

derived from [Json::Exception](#)

7.18.2 Constructor & Destructor Documentation

7.18.2.1 `Json::LogicError::LogicError (JSONCPP_STRING const & msg)`

7.18.3 Member Function Documentation

7.18.3.1 `char const * Json::Exception::what () const throw ()` `[inherited]`

7.18.4 Member Data Documentation

7.18.4.1 `JSONCPP_STRING Json::Exception::msg_` `[protected]`, `[inherited]`

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

- `/home/pranav/Repositories/zcm/include/json.hpp`
- `/home/pranav/Repositories/zcm/src/json.cpp`

7.19 zcm::Operation_Queue Class Reference

[Operation_Queue](#) class.

```
#include <operation_queue.hpp>
```

Classes

- struct [PriorityOrdering](#)

Public Member Functions

- void `enqueue` (`Base_Operation` *`new_operation`)
- void `dequeue` ()
- bool `empty` ()
- `Base_Operation` * `top` ()
- void `process` ()
- `std::thread` * `spawn` ()

Private Attributes

- `std::priority_queue`< `Base_Operation`, `std::vector`< `Base_Operation` * >, `PriorityOrdering` > `operation_queue`
The component operation queue - STL priority_queue with fixed-priority scheduling.
- `std::mutex` `queue_mutex`
Mutex that protects the queue during enqueue/dequeue.

7.19.1 Detailed Description

`Operation_Queue` class.

7.19.2 Member Function Documentation

7.19.2.1 void `zcm::Operation_Queue::dequeue` ()

7.19.2.2 bool `zcm::Operation_Queue::empty` ()

7.19.2.3 void `zcm::Operation_Queue::enqueue` (`Base_Operation` * `new_operation`)

7.19.2.4 void `zcm::Operation_Queue::process` ()

7.19.2.5 `std::thread` * `zcm::Operation_Queue::spawn` ()

7.19.2.6 `Base_Operation` * `zcm::Operation_Queue::top` ()

7.19.3 Member Data Documentation

7.19.3.1 `std::priority_queue`<`Base_Operation`, `std::vector`<`Base_Operation`*>, `PriorityOrdering`>
`zcm::Operation_Queue::operation_queue` [private]

The component operation queue - STL priority_queue with fixed-priority scheduling.

7.19.3.2 `std::mutex zcm::Operation_Queue::queue_mutex` `[private]`

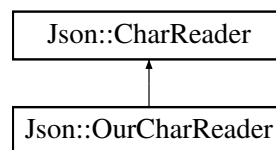
Mutex that protects the queue during enqueue/dequeue.

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

- [/home/pranav/Repositories/zcm/include/operation_queue.hpp](#)
- [/home/pranav/Repositories/zcm/src/operation_queue.cpp](#)

7.20 Json::OurCharReader Class Reference

Inheritance diagram for Json::OurCharReader:



Public Member Functions

- [OurCharReader](#) (bool collectComments, [OurFeatures](#) const &features)
- bool [parse](#) (char const *beginDoc, char const *endDoc, [Value](#) *root, [JSONCPP_STRING](#) *errs) [JSONCPP↔P_OVERRIDE](#)
Read a [Value](#) from a [JSON](#) document.

Private Attributes

- bool const [collectComments_](#)
- [OurReader](#) [reader_](#)

7.20.1 Constructor & Destructor Documentation

7.20.1.1 `Json::OurCharReader::OurCharReader (bool collectComments, OurFeatures const & features)` `[inline]`

7.20.2 Member Function Documentation

7.20.2.1 `bool Json::OurCharReader::parse (char const * beginDoc, char const * endDoc, Value * root, JSONCPP_STRING * errs)` `[inline], [virtual]`

Read a [Value](#) from a [JSON](#) document.

The document must be a UTF-8 encoded string containing the document to read.

Parameters

<i>beginDoc</i>	Pointer on the beginning of the UTF-8 encoded string of the document to read.
<i>endDoc</i>	Pointer on the end of the UTF-8 encoded string of the document to read. Must be \geq beginDoc.
<i>root</i>	[out] Contains the root value of the document if it was successfully parsed.
<i>errs</i>	[out] Formatted error messages (if not NULL) a user friendly string that lists errors in the parsed document.

Returns

`true` if the document was successfully parsed, `false` if an error occurred.

Implements [Json::CharReader](#).

7.20.3 Member Data Documentation

7.20.3.1 `bool const Json::OurCharReader::collectComments_` [private]

7.20.3.2 `OurReader Json::OurCharReader::reader_` [private]

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

- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.21 Json::OurFeatures Class Reference**Static Public Member Functions**

- static [OurFeatures all](#) ()

Public Attributes

- bool [allowComments_](#)
- bool [strictRoot_](#)
- bool [allowDroppedNullPlaceholders_](#)
- bool [allowNumericKeys_](#)
- bool [allowSingleQuotes_](#)
- bool [faillfExtra_](#)
- bool [rejectDupKeys_](#)
- bool [allowSpecialFloats_](#)
- int [stackLimit_](#)

7.21.1 Member Function Documentation

7.21.1.1 `OurFeatures` `Json::OurFeatures::all ()` `[static]`

7.21.2 Member Data Documentation

7.21.2.1 `bool` `Json::OurFeatures::allowComments_`

7.21.2.2 `bool` `Json::OurFeatures::allowDroppedNullPlaceholders_`

7.21.2.3 `bool` `Json::OurFeatures::allowNumericKeys_`

7.21.2.4 `bool` `Json::OurFeatures::allowSingleQuotes_`

7.21.2.5 `bool` `Json::OurFeatures::allowSpecialFloats_`

7.21.2.6 `bool` `Json::OurFeatures::failIfExtra_`

7.21.2.7 `bool` `Json::OurFeatures::rejectDupKeys_`

7.21.2.8 `int` `Json::OurFeatures::stackLimit_`

7.21.2.9 `bool` `Json::OurFeatures::strictRoot_`

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

- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.22 Json::OurReader Class Reference

Classes

- class [ErrorInfo](#)
- struct [StructuredError](#)
- class [Token](#)

Public Types

- typedef char [Char](#)
- typedef const [Char](#) * [Location](#)

Public Member Functions

- [OurReader](#) ([OurFeatures](#) const &features)
- bool [parse](#) (const char *beginDoc, const char *endDoc, [Value](#) &root, bool collectComments=true)
- [JSONCPP_STRING](#) [getFormattedErrorMessages](#) () const
- std::vector< [StructuredError](#) > [getStructuredErrors](#) () const
- bool [pushError](#) (const [Value](#) &value, const [JSONCPP_STRING](#) &message)
- bool [pushError](#) (const [Value](#) &value, const [JSONCPP_STRING](#) &message, const [Value](#) &extra)
- bool [good](#) () const

Private Types

- enum [TokenType](#) {
[tokenEndOfStream](#) = 0, [tokenObjectBegin](#), [tokenObjectEnd](#), [tokenArrayBegin](#),
[tokenArrayEnd](#), [tokenString](#), [tokenNumber](#), [tokenTrue](#),
[tokenFalse](#), [tokenNull](#), [tokenNaN](#), [tokenPosInf](#),
[tokenNegInf](#), [tokenArraySeparator](#), [tokenMemberSeparator](#), [tokenComment](#),
[tokenError](#) }
- typedef std::deque< [ErrorInfo](#) > [Errors](#)
- typedef std::stack< [Value](#) * > [Nodes](#)

Private Member Functions

- [OurReader](#) ([OurReader](#) const &)
- void [operator=](#) ([OurReader](#) const &)
- bool [readToken](#) ([Token](#) &token)
- void [skipSpaces](#) ()
- bool [match](#) ([Location](#) pattern, int patternLength)
- bool [readComment](#) ()
- bool [readCStyleComment](#) ()
- bool [readCppStyleComment](#) ()
- bool [readString](#) ()
- bool [readStringSingleQuote](#) ()
- bool [readNumber](#) (bool checkInf)
- bool [readValue](#) ()
- bool [readObject](#) ([Token](#) &token)
- bool [readArray](#) ([Token](#) &token)
- bool [decodeNumber](#) ([Token](#) &token)
- bool [decodeNumber](#) ([Token](#) &token, [Value](#) &decoded)
- bool [decodeString](#) ([Token](#) &token)
- bool [decodeString](#) ([Token](#) &token, [JSONCPP_STRING](#) &decoded)
- bool [decodeDouble](#) ([Token](#) &token)
- bool [decodeDouble](#) ([Token](#) &token, [Value](#) &decoded)
- bool [decodeUnicodeCodePoint](#) ([Token](#) &token, [Location](#) ¤t, [Location](#) end, unsigned int &unicode)
- bool [decodeUnicodeEscapeSequence](#) ([Token](#) &token, [Location](#) ¤t, [Location](#) end, unsigned int &unicode)
- bool [addError](#) (const [JSONCPP_STRING](#) &message, [Token](#) &token, [Location](#) extra=0)
- bool [recoverFromError](#) ([TokenType](#) skipUntilToken)
- bool [addErrorAndRecover](#) (const [JSONCPP_STRING](#) &message, [Token](#) &token, [TokenType](#) skipUntilToken)
- void [skipUntilSpace](#) ()
- [Value](#) & [currentValue](#) ()
- [Char](#) [getNextChar](#) ()
- void [getLocationLineAndColumn](#) ([Location](#) location, int &line, int &column) const
- [JSONCPP_STRING](#) [getLocationLineAndColumn](#) ([Location](#) location) const
- void [addComment](#) ([Location](#) begin, [Location](#) end, [CommentPlacement](#) placement)
- void [skipCommentTokens](#) ([Token](#) &token)

Private Attributes

- [Nodes](#) `nodes_`
- [Errors](#) `errors_`
- [JSONCPP_STRING](#) `document_`
- [Location](#) `begin_`
- [Location](#) `end_`
- [Location](#) `current_`
- [Location](#) `lastValueEnd_`
- [Value](#) * `lastValue_`
- [JSONCPP_STRING](#) `commentsBefore_`
- `int` `stackDepth_`
- [OurFeatures](#) `const features_`
- `bool` `collectComments_`

7.22.1 Member Typedef Documentation

7.22.1.1 `typedef char Json::OurReader::Char`

7.22.1.2 `typedef std::deque<ErrorInfo> Json::OurReader::Errors` `[private]`

7.22.1.3 `typedef const Char* Json::OurReader::Location`

7.22.1.4 `typedef std::stack<Value*> Json::OurReader::Nodes` `[private]`

7.22.2 Member Enumeration Documentation

7.22.2.1 `enum Json::OurReader::TokenType` `[private]`

Enumerator

tokenEndOfStream
tokenObjectBegin
tokenObjectEnd
tokenArrayBegin
tokenArrayEnd
tokenString
tokenNumber
tokenTrue
tokenFalse
tokenNull
tokenNaN
tokenPosInf
tokenNegInf
tokenArraySeparator
tokenMemberSeparator
tokenComment
tokenError

7.22.3 Constructor & Destructor Documentation

7.22.3.1 `Json::OurReader::OurReader (OurFeatures const & features)`

7.22.3.2 `Json::OurReader::OurReader (OurReader const &) [private]`

7.22.4 Member Function Documentation

7.22.4.1 `void Json::OurReader::addComment (Location begin, Location end, CommentPlacement placement) [private]`

7.22.4.2 `bool Json::OurReader::addError (const JSONCPP_STRING & message, Token & token, Location extra = 0) [private]`

7.22.4.3 `bool Json::OurReader::addErrorAndRecover (const JSONCPP_STRING & message, Token & token, TokenType skipUntilToken) [private]`

7.22.4.4 `Value & Json::OurReader::currentValue () [private]`

7.22.4.5 `bool Json::OurReader::decodeDouble (Token & token) [private]`

7.22.4.6 `bool Json::OurReader::decodeDouble (Token & token, Value & decoded) [private]`

7.22.4.7 `bool Json::OurReader::decodeNumber (Token & token) [private]`

7.22.4.8 `bool Json::OurReader::decodeNumber (Token & token, Value & decoded) [private]`

7.22.4.9 `bool Json::OurReader::decodeString (Token & token) [private]`

7.22.4.10 `bool Json::OurReader::decodeString (Token & token, JSONCPP_STRING & decoded) [private]`

7.22.4.11 `bool Json::OurReader::decodeUnicodeCodePoint (Token & token, Location & current, Location end, unsigned int & unicode) [private]`

7.22.4.12 `bool Json::OurReader::decodeUnicodeEscapeSequence (Token & token, Location & current, Location end, unsigned int & unicode) [private]`

7.22.4.13 `JSONCPP_STRING Json::OurReader::getFormattedErrorMessages () const`

7.22.4.14 `void Json::OurReader::getLocationLineAndColumn (Location location, int & line, int & column) const [private]`

7.22.4.15 `JSONCPP_STRING Json::OurReader::getLocationLineAndColumn (Location location) const [private]`

7.22.4.16 `OurReader::Char Json::OurReader::getNextChar () [private]`

- 7.22.4.17 `std::vector< OurReader::StructuredError > Json::OurReader::getStructuredErrors () const`
- 7.22.4.18 `bool Json::OurReader::good () const`
- 7.22.4.19 `bool Json::OurReader::match (Location pattern, int patternLength) [private]`
- 7.22.4.20 `void Json::OurReader::operator= (OurReader const &) [private]`
- 7.22.4.21 `bool Json::OurReader::parse (const char * beginDoc, const char * endDoc, Value & root, bool collectComments = true)`
- 7.22.4.22 `bool Json::OurReader::pushError (const Value & value, const JSONCPP_STRING & message)`
- 7.22.4.23 `bool Json::OurReader::pushError (const Value & value, const JSONCPP_STRING & message, const Value & extra)`
- 7.22.4.24 `bool Json::OurReader::readArray (Token & token) [private]`
- 7.22.4.25 `bool Json::OurReader::readComment () [private]`
- 7.22.4.26 `bool Json::OurReader::readCppStyleComment () [private]`
- 7.22.4.27 `bool Json::OurReader::readCStyleComment () [private]`
- 7.22.4.28 `bool Json::OurReader::readNumber (bool checkInf) [private]`
- 7.22.4.29 `bool Json::OurReader::readObject (Token & token) [private]`
- 7.22.4.30 `bool Json::OurReader::readString () [private]`
- 7.22.4.31 `bool Json::OurReader::readStringSingleQuote () [private]`
- 7.22.4.32 `bool Json::OurReader::readToken (Token & token) [private]`
- 7.22.4.33 `bool Json::OurReader::readValue () [private]`
- 7.22.4.34 `bool Json::OurReader::recoverFromError (TokenType skipUntilToken) [private]`
- 7.22.4.35 `void Json::OurReader::skipCommentTokens (Token & token) [private]`
- 7.22.4.36 `void Json::OurReader::skipSpaces () [private]`
- 7.22.4.37 `void Json::OurReader::skipUntilSpace () [private]`

7.22.5 Member Data Documentation

- 7.22.5.1 **Location** `Json::OurReader::begin_` [private]
- 7.22.5.2 **bool** `Json::OurReader::collectComments_` [private]
- 7.22.5.3 **JSONCPP_STRING** `Json::OurReader::commentsBefore_` [private]
- 7.22.5.4 **Location** `Json::OurReader::current_` [private]
- 7.22.5.5 **JSONCPP_STRING** `Json::OurReader::document_` [private]
- 7.22.5.6 **Location** `Json::OurReader::end_` [private]
- 7.22.5.7 **Errors** `Json::OurReader::errors_` [private]
- 7.22.5.8 **OurFeatures** `const Json::OurReader::features_` [private]
- 7.22.5.9 **Value*** `Json::OurReader::lastValue_` [private]
- 7.22.5.10 **Location** `Json::OurReader::lastValueEnd_` [private]
- 7.22.5.11 **Nodes** `Json::OurReader::nodes_` [private]
- 7.22.5.12 **int** `Json::OurReader::stackDepth_` [private]

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

- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.23 Json::Path Class Reference

Experimental and untested: represents a "path" to access a node.

```
#include <json.hpp>
```

Public Member Functions

- **Path** (`const JSONCPP_STRING &path`, `const PathArgument &a1=PathArgument()`, `const PathArgument &a2=PathArgument()`, `const PathArgument &a3=PathArgument()`, `const PathArgument &a4=PathArgument()`, `const PathArgument &a5=PathArgument()`)
- `const Value &resolve` (`const Value &root`) `const`
- `Value resolve` (`const Value &root`, `const Value &defaultValue`) `const`
- `Value &make` (`Value &root`) `const`

Creates the "path" to access the specified node and returns a reference on the node.

Private Types

- typedef std::vector< const [PathArgument](#) * > [InArgs](#)
- typedef std::vector< [PathArgument](#) > [Args](#)

Private Member Functions

- void [makePath](#) (const [JSONCPP_STRING](#) &path, const [InArgs](#) &in)
- void [addPathInArg](#) (const [JSONCPP_STRING](#) &path, const [InArgs](#) &in, [InArgs::const_iterator](#) &itInArg, [PathArgument::Kind](#) kind)
- void [invalidPath](#) (const [JSONCPP_STRING](#) &path, int location)

Private Attributes

- [Args](#) [args_](#)

7.23.1 Detailed Description

Experimental and untested: represents a "path" to access a node.

Syntax:

- "." => root node
- "[n]" => elements at index 'n' of root node (an array value)
- ".name" => member named 'name' of root node (an object value)
- ".name1.name2.name3"
- ".[0][1][2].name1[3]"
- ".%" => member name is provided as parameter
- ".[%]" => index is provided as parameter

7.23.2 Member Typedef Documentation

7.23.2.1 typedef std::vector<[PathArgument](#)> [Json::Path::Args](#) [private]

7.23.2.2 typedef std::vector<const [PathArgument](#)*> [Json::Path::InArgs](#) [private]

7.23.3 Constructor & Destructor Documentation

7.23.3.1 [Json::Path::Path](#) (const [JSONCPP_STRING](#) & *path*, const [PathArgument](#) & *a1* = [PathArgument](#) (), const [PathArgument](#) & *a2* = [PathArgument](#) (), const [PathArgument](#) & *a3* = [PathArgument](#) (), const [PathArgument](#) & *a4* = [PathArgument](#) (), const [PathArgument](#) & *a5* = [PathArgument](#) ())

7.23.4 Member Function Documentation

7.23.4.1 void [Json::Path::addPathInArg](#) (const [JSONCPP_STRING](#) & *path*, const [InArgs](#) & *in*, [InArgs::const_iterator](#) & *itInArg*, [PathArgument::Kind](#) *kind*) [private]

7.23.4.2 void [Json::Path::invalidPath](#) (const [JSONCPP_STRING](#) & *path*, int *location*) [private]

7.23.4.3 Value & [Json::Path::make](#) (Value & *root*) const

Creates the "path" to access the specified node and returns a reference on the node.

7.23.4.4 `void Json::Path::makePath (const JSONCPP_STRING & path, const InArgs & in)` `[private]`

7.23.4.5 `const Value & Json::Path::resolve (const Value & root) const`

7.23.4.6 `Value Json::Path::resolve (const Value & root, const Value & defaultValue) const`

7.23.5 Member Data Documentation

7.23.5.1 `Args Json::Path::args_` `[private]`

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.24 Json::PathArgument Class Reference

Experimental and untested: represents an element of the "path" to access a node.

```
#include <json.hpp>
```

Public Member Functions

- [PathArgument](#) ()
- [PathArgument](#) ([ArrayIndex](#) index)
- [PathArgument](#) (const char *key)
- [PathArgument](#) (const JSONCPP_STRING &key)

Private Types

- enum [Kind](#) { [kindNone](#) = 0, [kindIndex](#), [kindKey](#) }

Private Attributes

- [JSONCPP_STRING](#) [key_](#)
- [ArrayIndex](#) [index_](#)
- [Kind](#) [kind_](#)

Friends

- class [Path](#)

7.24.1 Detailed Description

Experimental and untested: represents an element of the "path" to access a node.

7.24.2 Member Enumeration Documentation

7.24.2.1 enum `Json::PathArgument::Kind` `[private]`

Enumerator

kindNone

kindIndex

kindKey

7.24.3 Constructor & Destructor Documentation

7.24.3.1 `Json::PathArgument::PathArgument ()`

7.24.3.2 `Json::PathArgument::PathArgument (ArrayIndex index)`

7.24.3.3 `Json::PathArgument::PathArgument (const char * key)`

7.24.3.4 `Json::PathArgument::PathArgument (const JSONCPP_STRING & key)`

7.24.4 Friends And Related Function Documentation

7.24.4.1 `friend class Path` `[friend]`

7.24.5 Member Data Documentation

7.24.5.1 `ArrayIndex Json::PathArgument::index_` `[private]`

7.24.5.2 `JSONCPP_STRING Json::PathArgument::key_` `[private]`

7.24.5.3 `Kind Json::PathArgument::kind_` `[private]`

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.25 zcm::Operation_Queue::PriorityOrdering Struct Reference

```
#include <operation_queue.hpp>
```

Public Member Functions

- `bool operator() (const Base_Operation *lhs, const Base_Operation *rhs) const`

7.25.1 Member Function Documentation

7.25.1.1 `bool zcm::Operation_Queue::PriorityOrdering::operator() (const Base_Operation * lhs, const Base_Operation * rhs) const` `[inline]`

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

- [/home/pranav/Repositories/zcm/include/operation_queue.hpp](#)

7.26 zcm::Publisher Class Reference

[Publisher](#) class.

```
#include <publisher.hpp>
```

Public Member Functions

- [Publisher](#) (std::string [name](#))
Construct a publisher object.
- [Publisher](#) (std::string [name](#), std::vector< std::string > [endpoints](#))
Construct a publisher object with known endpoints.
- [~Publisher](#) ()
Close the publisher ZMQ socket and destroy the context.
- void [bind](#) (std::vector< std::string > [new_endpoints](#))
Bind the publisher to a new set of endpoints.
- std::string [get_name](#) ()
Return the publisher name.
- void [add_connection](#) (std::string [new_connection](#))
Add a new endpoint to the publisher.
- void [send](#) (std::string [message](#))
Publish a new message.

Private Attributes

- std::string [name](#)
Name of the publisher.
- zmq::context_t * [context](#)
ZMQ Context of the publisher.
- zmq::socket_t * [publisher_socket](#)
ZMQ Socket of the publisher.
- std::vector< std::string > [endpoints](#)
Vector of endpoints to bind to.

7.26.1 Detailed Description

[Publisher](#) class.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 `zcm::Publisher::Publisher (std::string name)`

Construct a publisher object.

Parameters

in	<i>name</i>	Publisher name
----	-------------	--------------------------------

7.26.2.2 `zcm::Publisher::Publisher (std::string name, std::vector< std::string > endpoints)`

Construct a publisher object with known endpoints.

Parameters

in	<i>name</i>	Publisher name
in	<i>endpoints</i>	A vector of endpoint strings

7.26.2.3 `zcm::Publisher::~~Publisher ()`

Close the publisher ZMQ socket and destroy the context.

7.26.3 Member Function Documentation

7.26.3.1 `void zcm::Publisher::add_connection (std::string new_connection)`

Add a new endpoint to the publisher.

Parameters

in	<i>new_connection</i>	New endpoint to bind to
----	-----------------------	-------------------------

7.26.3.2 `void zcm::Publisher::bind (std::vector< std::string > new_endpoints)`

Bind the publisher to a new set of endpoints.

Parameters

in	<i>new_endpoints</i>	New set of endpoints as a vector
----	----------------------	----------------------------------

7.26.3.3 `std::string zcm::Publisher::get_name ()`

Return the publisher name.

Returns

[Publisher](#) name

7.26.3.4 void zcm::Publisher::send (std::string *message*)

Publish a new message.

Parameters

in	<i>message</i>	The message string. Serialize complex objects to strings with protobuf
----	----------------	--

7.26.4 Member Data Documentation

7.26.4.1 zmq::context_t* zcm::Publisher::context [private]

ZMQ Context of the publisher.

7.26.4.2 std::vector<std::string> zcm::Publisher::endpoints [private]

Vector of endpoints to bind to.

7.26.4.3 std::string zcm::Publisher::name [private]

Name of the publisher.

7.26.4.4 zmq::socket_t* zcm::Publisher::publisher_socket [private]

ZMQ Socket of the publisher.

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

- [/home/pranav/Repositories/zcm/include/publisher.hpp](#)
- [/home/pranav/Repositories/zcm/src/publisher.cpp](#)

7.27 Json::Reader Class Reference

Unserialize a [JSON](#) document into a [Value](#).

```
#include <json.hpp>
```

Classes

- class [ErrorInfo](#)
- struct [StructuredError](#)
 - An error tagged with where in the JSON text it was encountered.*
- class [Token](#)

Public Types

- typedef char [Char](#)
- typedef const [Char](#) * [Location](#)

Public Member Functions

- [Reader](#) ()
Constructs a [Reader](#) allowing all features for parsing.
- [Reader](#) (const [Features](#) &features)
Constructs a [Reader](#) allowing the specified feature set for parsing.
- bool [parse](#) (const std::string &document, [Value](#) &root, bool collectComments=true)
Read a [Value](#) from a [JSON](#) document.
- bool [parse](#) (const char *beginDoc, const char *endDoc, [Value](#) &root, bool collectComments=true)
Read a [Value](#) from a [JSON](#) document.
- bool [parse](#) ([JSONCPP_ISTREAM](#) &is, [Value](#) &root, bool collectComments=true)
Parse from input stream.
- [JSONCPP_STRING](#) [getFormattedErrorMessages](#) () const
Returns a user friendly string that list errors in the parsed document.
- [JSONCPP_STRING](#) [getFormattedErrorMessages](#) () const
Returns a user friendly string that list errors in the parsed document.
- std::vector< [StructuredError](#) > [getStructuredErrors](#) () const
Returns a vector of structured errors encountered while parsing.
- bool [pushError](#) (const [Value](#) &value, const [JSONCPP_STRING](#) &message)
Add a semantic error message.
- bool [pushError](#) (const [Value](#) &value, const [JSONCPP_STRING](#) &message, const [Value](#) &extra)
Add a semantic error message with extra context.
- bool [good](#) () const
Return whether there are any errors.

Private Types

- enum [TokenType](#) {
 [tokenEndOfStream](#) = 0, [tokenObjectBegin](#), [tokenObjectEnd](#), [tokenArrayBegin](#),
 [tokenArrayEnd](#), [tokenString](#), [tokenNumber](#), [tokenTrue](#),
 [tokenFalse](#), [tokenNull](#), [tokenArraySeparator](#), [tokenMemberSeparator](#),
 [tokenComment](#), [tokenError](#) }
- typedef std::deque< [ErrorInfo](#) > [Errors](#)
- typedef std::stack< [Value](#) * > [Nodes](#)

Private Member Functions

- bool [readToken](#) ([Token](#) &token)
- void [skipSpaces](#) ()
- bool [match](#) ([Location](#) pattern, int patternLength)
- bool [readComment](#) ()
- bool [readCStyleComment](#) ()
- bool [readCppStyleComment](#) ()
- bool [readString](#) ()
- void [readNumber](#) ()

- bool [readValue](#) ()
- bool [readObject](#) ([Token](#) &token)
- bool [readArray](#) ([Token](#) &token)
- bool [decodeNumber](#) ([Token](#) &token)
- bool [decodeNumber](#) ([Token](#) &token, [Value](#) &decoded)
- bool [decodeString](#) ([Token](#) &token)
- bool [decodeString](#) ([Token](#) &token, [JSONCPP_STRING](#) &decoded)
- bool [decodeDouble](#) ([Token](#) &token)
- bool [decodeDouble](#) ([Token](#) &token, [Value](#) &decoded)
- bool [decodeUnicodeCodePoint](#) ([Token](#) &token, [Location](#) ¤t, [Location](#) end, unsigned int &unicode)
- bool [decodeUnicodeEscapeSequence](#) ([Token](#) &token, [Location](#) ¤t, [Location](#) end, unsigned int &unicode)
- bool [addError](#) (const [JSONCPP_STRING](#) &message, [Token](#) &token, [Location](#) extra=0)
- bool [recoverFromError](#) ([TokenType](#) skipUntilToken)
- bool [addErrorAndRecover](#) (const [JSONCPP_STRING](#) &message, [Token](#) &token, [TokenType](#) skipUntilToken)
- void [skipUntilSpace](#) ()
- [Value](#) & [currentValue](#) ()
- [Char](#) [getNextChar](#) ()
- void [getLocationLineAndColumn](#) ([Location](#) location, int &line, int &column) const
- [JSONCPP_STRING](#) [getLocationLineAndColumn](#) ([Location](#) location) const
- void [addComment](#) ([Location](#) begin, [Location](#) end, [CommentPlacement](#) placement)
- void [skipCommentTokens](#) ([Token](#) &token)

Private Attributes

- [Nodes](#) [nodes_](#)
- [Errors](#) [errors_](#)
- [JSONCPP_STRING](#) [document_](#)
- [Location](#) [begin_](#)
- [Location](#) [end_](#)
- [Location](#) [current_](#)
- [Location](#) [lastValueEnd_](#)
- [Value](#) * [lastValue_](#)
- [JSONCPP_STRING](#) [commentsBefore_](#)
- [Features](#) [features_](#)
- bool [collectComments_](#)

7.27.1 Detailed Description

Unserialize a [JSON](#) document into a [Value](#).

Deprecated Use [CharReader](#) and [CharReaderBuilder](#).

7.27.2 Member Typedef Documentation

7.27.2.1 `typedef char Json::Reader::Char`

7.27.2.2 `typedef std::deque<ErrorInfo> Json::Reader::Errors` [private]

7.27.2.3 `typedef const Char* Json::Reader::Location`

7.27.2.4 `typedef std::stack<Value*> Json::Reader::Nodes` [private]

7.27.3 Member Enumeration Documentation

7.27.3.1 `enum Json::Reader::TokenType` [private]

Enumerator

tokenEndOfStream

tokenObjectBegin

tokenObjectEnd

tokenArrayBegin

tokenArrayEnd

tokenString

tokenNumber

tokenTrue

tokenFalse

tokenNull

tokenArraySeparator

tokenMemberSeparator

tokenComment

tokenError

7.27.4 Constructor & Destructor Documentation

7.27.4.1 `Json::Reader::Reader ()`

Constructs a [Reader](#) allowing all features for parsing.

7.27.4.2 `Json::Reader::Reader (const Features & features)`

Constructs a [Reader](#) allowing the specified feature set for parsing.

7.27.5 Member Function Documentation

7.27.5.1 `void Json::Reader::addComment (Location begin, Location end, CommentPlacement placement)`
[private]

7.27.5.2 `bool Json::Reader::addError (const JSONCPP_STRING & message, Token & token, Location extra = 0)`
[private]

7.27.5.3 `bool Json::Reader::addErrorAndRecover (const JSONCPP_STRING & message, Token & token, TokenType skipUntilToken)` [private]

7.27.5.4 `Value & Json::Reader::currentValue ()` [private]

7.27.5.5 `bool Json::Reader::decodeDouble (Token & token)` [private]

7.27.5.6 `bool Json::Reader::decodeDouble (Token & token, Value & decoded)` [private]

7.27.5.7 `bool Json::Reader::decodeNumber (Token & token)` [private]

7.27.5.8 `bool Json::Reader::decodeNumber (Token & token, Value & decoded)` [private]

7.27.5.9 `bool Json::Reader::decodeString (Token & token)` [private]

7.27.5.10 `bool Json::Reader::decodeString (Token & token, JSONCPP_STRING & decoded)` [private]

7.27.5.11 `bool Json::Reader::decodeUnicodeCodePoint (Token & token, Location & current, Location end, unsigned int & unicode)` [private]

7.27.5.12 `bool Json::Reader::decodeUnicodeEscapeSequence (Token & token, Location & current, Location end, unsigned int & unicode)` [private]

7.27.5.13 `JSONCPP_STRING Json::Reader::getFormattedErrorMessages () const`

Returns a user friendly string that list errors in the parsed document.

Returns

Formatted error message with the list of errors with their location in the parsed document. An empty string is returned if no error occurred during parsing.

Deprecated Use [getFormattedErrorMessages\(\)](#) instead (typo fix).

7.27.5.14 `JSONCPP_STRING Json::Reader::getFormattedErrorMessages () const`

Returns a user friendly string that list errors in the parsed document.

Returns

Formatted error message with the list of errors with their location in the parsed document. An empty string is returned if no error occurred during parsing.

7.27.5.15 `void Json::Reader::getLocationLineAndColumn (Location location, int & line, int & column) const`
`[private]`

7.27.5.16 `JSONCPP_STRING Json::Reader::getLocationLineAndColumn (Location location) const` `[private]`

7.27.5.17 `Reader::Char Json::Reader::getNextChar ()` `[private]`

7.27.5.18 `std::vector< Reader::StructuredError > Json::Reader::getStructuredErrors () const`

Returns a vector of structured errors encountered while parsing.

Returns

A (possibly empty) vector of [StructuredError](#) objects. Currently only one error can be returned, but the caller should tolerate multiple errors. This can occur if the parser recovers from a non-fatal parse error and then encounters additional errors.

7.27.5.19 `bool Json::Reader::good () const`

Return whether there are any errors.

Returns

`true` if there are no errors to report `false` if errors have occurred.

7.27.5.20 `bool Json::Reader::match (Location pattern, int patternLength)` `[private]`

7.27.5.21 `bool Json::Reader::parse (const std::string & document, Value & root, bool collectComments = true)`

Read a [Value](#) from a [JSON](#) document.

Parameters

<i>document</i>	UTF-8 encoded string containing the document to read.
<i>root</i>	[out] Contains the root value of the document if it was successfully parsed.
<i>collectComments</i>	<code>true</code> to collect comment and allow writing them back during serialization, <code>false</code> to discard comments. This parameter is ignored if Features::allowComments_ is <code>false</code> .

Returns

`true` if the document was successfully parsed, `false` if an error occurred.

7.27.5.22 `bool Json::Reader::parse (const char * beginDoc, const char * endDoc, Value & root, bool collectComments = true)`

Read a [Value](#) from a [JSON](#) document.

Parameters

<i>beginDoc</i>	Pointer on the beginning of the UTF-8 encoded string of the document to read.
<i>endDoc</i>	Pointer on the end of the UTF-8 encoded string of the document to read. Must be <code>>= beginDoc</code> .
<i>root</i>	[out] Contains the root value of the document if it was successfully parsed.
<i>collectComments</i>	<code>true</code> to collect comment and allow writing them back during serialization, <code>false</code> to discard comments. This parameter is ignored if <code>Features::allowComments_</code> is <code>false</code> .

Returns

`true` if the document was successfully parsed, `false` if an error occurred.

7.27.5.23 `bool Json::Reader::parse (JSONCPP_ISTREAM & is, Value & root, bool collectComments = true)`

Parse from input stream.

See also

`Json::operator>>(std::istream&, Json::Value&).`

7.27.5.24 `bool Json::Reader::pushError (const Value & value, const JSONCPP_STRING & message)`

Add a semantic error message.

Parameters

<i>value</i>	JSON Value location associated with the error
<i>message</i>	The error message.

Returns

`true` if the error was successfully added, `false` if the [Value](#) offset exceeds the document size.

7.27.5.25 `bool Json::Reader::pushError (const Value & value, const JSONCPP_STRING & message, const Value & extra)`

Add a semantic error message with extra context.

Parameters

<i>value</i>	JSON Value location associated with the error
<i>message</i>	The error message.
<i>extra</i>	Additional JSON Value location to contextualize the error

Returns

`true` if the error was successfully added, `false` if either `Value` offset exceeds the document size.

- 7.27.5.26 `bool Json::Reader::readArray (Token & token)` [private]
- 7.27.5.27 `bool Json::Reader::readComment ()` [private]
- 7.27.5.28 `bool Json::Reader::readCppStyleComment ()` [private]
- 7.27.5.29 `bool Json::Reader::readCStyleComment ()` [private]
- 7.27.5.30 `void Json::Reader::readNumber ()` [private]
- 7.27.5.31 `bool Json::Reader::readObject (Token & token)` [private]
- 7.27.5.32 `bool Json::Reader::readString ()` [private]
- 7.27.5.33 `bool Json::Reader::readToken (Token & token)` [private]
- 7.27.5.34 `bool Json::Reader::readValue ()` [private]
- 7.27.5.35 `bool Json::Reader::recoverFromError (TokenType skipUntilToken)` [private]
- 7.27.5.36 `void Json::Reader::skipCommentTokens (Token & token)` [private]
- 7.27.5.37 `void Json::Reader::skipSpaces ()` [private]
- 7.27.5.38 `void Json::Reader::skipUntilSpace ()` [private]

7.27.6 Member Data Documentation

- 7.27.6.1 `Location Json::Reader::begin_` [private]
- 7.27.6.2 `bool Json::Reader::collectComments_` [private]
- 7.27.6.3 `JSONCPP_STRING Json::Reader::commentsBefore_` [private]
- 7.27.6.4 `Location Json::Reader::current_` [private]
- 7.27.6.5 `JSONCPP_STRING Json::Reader::document_` [private]
- 7.27.6.6 `Location Json::Reader::end_` [private]
- 7.27.6.7 `Errors Json::Reader::errors_` [private]
- 7.27.6.8 `Features Json::Reader::features_` [private]
- 7.27.6.9 `Value* Json::Reader::lastValue_` [private]
- 7.27.6.10 `Location Json::Reader::lastValueEnd_` [private]
- 7.27.6.11 `Nodes Json::Reader::nodes_` [private]

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

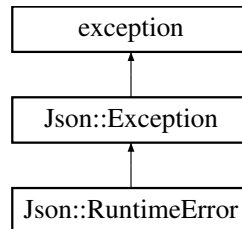
- `/home/pranav/Repositories/zcm/include/json.hpp`
- `/home/pranav/Repositories/zcm/src/json.cpp`

7.28 Json::RuntimeError Class Reference

Exceptions which the user cannot easily avoid.

```
#include <json.hpp>
```

Inheritance diagram for Json::RuntimeError:



Public Member Functions

- [RuntimeError](#) ([JSONCPP_STRING](#) const &msg)
- char const * [what](#) () const [JSONCPP_OVERRIDE](#) throw ()

Protected Attributes

- [JSONCPP_STRING](#) msg_

7.28.1 Detailed Description

Exceptions which the user cannot easily avoid.

E.g. out-of-memory (when we use malloc), stack-overflow, malicious input

Remarks

derived from [Json::Exception](#)

7.28.2 Constructor & Destructor Documentation

7.28.2.1 [Json::RuntimeError::RuntimeError](#) ([JSONCPP_STRING](#) const & msg)

7.28.3 Member Function Documentation

7.28.3.1 char const * [Json::Exception::what](#) () const throw) [\[inherited\]](#)

7.28.4 Member Data Documentation

7.28.4.1 [JSONCPP_STRING](#) [Json::Exception::msg_](#) [\[protected\]](#), [\[inherited\]](#)

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.29 zcm::Server Class Reference

[Server](#) class.

```
#include <server.hpp>
```

Public Member Functions

- [Server](#) (std::string [name](#), unsigned int [priority](#), std::function< std::string(const std::string &)> [operation_](#)↵
[function](#), [Operation_Queue](#) *[operation_queue_ptr](#))
Construct a server object.
- [Server](#) (std::string [name](#), unsigned int [priority](#), std::vector< std::string > [endpoints](#), std::function< std↵
::string(const std::string &)> [operation_function](#), [Operation_Queue](#) *[operation_queue_ptr](#))
Construct a server object with known endpoints.
- [~Server](#) ()
Close the server socket and destroy the ZMQ context.
- void [bind](#) (std::vector< std::string > [new_endpoints](#))
Bind to a new set of endpoints param[in] new_endpoints A new vector of endpoints to bind to.
- std::string [get_name](#) ()
Get the name of the server.
- unsigned int [get_priority](#) ()
Get the priority of the server.
- void [add_connection](#) (std::string [new_connection](#))
Add a new connection to the server.
- void [recv](#) ()
*Thread function of the server Behavior: (1) Wait for a new request on the server ZMQ socket (2) Create a [Server](#)
[Operation](#) (3) Enqueue onto operation_queue (4) Goto step (1)*
- void [rebind_operation_function](#) (std::function< std::string(const std::string &)> [new_operation_function](#))
Rebind the server operation function.
- std::thread [spawn](#) ()
Spawn a new thread for the server.
- void [start](#) ()
Start the server thread.

Private Attributes

- std::string [name](#)
Name of the server.
- unsigned int [priority](#)
Priority of the server.
- std::vector< std::string > [endpoints](#)
Vector of connection endpoints.
- std::function< std::string(const std::string &)> [operation_function](#)
Operation function bound to the server - [Component](#) method that handles received requests.
- [Operation_Queue](#) * [operation_queue_ptr](#)
Pointer to the operation_queue.
- zmq::context_t * [context](#)
Pointer to the server ZMQ context.
- zmq::socket_t * [server_socket](#)
Pointer to the server ZMQ socket.
- bool [ready](#)
Boolean representing the state of the server to receive new requests.
- std::mutex [func_mutex](#)
Mutex used when changing operation_function at runtime.

7.29.1 Detailed Description

[Server](#) class.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 `zcm::Server::Server (std::string name, unsigned int priority, std::function< std::string(const std::string &)> operation_function, Operation_Queue * operation_queue_ptr) [inline]`

Construct a server object.

Parameters

in	<i>name</i>	Server name
in	<i>priority</i>	Priority of the server
in	<i>operation_function</i>	Operation function of the server
in	<i>operation_queue_ptr</i>	Pointer to the operation queue

7.29.2.2 `zcm::Server::Server (std::string name, unsigned int priority, std::vector< std::string > endpoints, std::function< std::string(const std::string &)> operation_function, Operation_Queue * operation_queue_ptr)`

Construct a server object with known endpoints.

Parameters

in	<i>name</i>	Server name
in	<i>priority</i>	Priority of the server
in	<i>endpoints</i>	A vector of endpoints to bind to
in	<i>operation_function</i>	Operation function of the server
in	<i>operation_queue_ptr</i>	Pointer to the operation queue

7.29.2.3 `zcm::Server::~~Server ()`

Close the server socket and destroy the ZMQ context.

7.29.3 Member Function Documentation

7.29.3.1 `void zcm::Server::add_connection (std::string new_connection)`

Add a new connection to the server.

Parameters

in	<i>new_connection</i>	New connection address to bind to
----	-----------------------	-----------------------------------

7.29.3.2 `void zcm::Server::bind (std::vector< std::string > new_endpoints)`

Bind to a new set of endpoints param[in] *new_endpoints* A new vector of endpoints to bind to.

7.29.3.3 `std::string zcm::Server::get_name ()`

Get the name of the server.

7.29.3.4 `unsigned int zcm::Server::get_priority ()`

Get the priority of the server.

7.29.3.5 `void zcm::Server::rebind_operation_function (std::function< std::string(const std::string &)> new_operation_function)`

Rebind the server operation function.

Parameters

in	<i>new_operation_function</i>	New server function to be handled upon recv()
----	-------------------------------	---

7.29.3.6 `void zcm::Server::recv ()`

Thread function of the server Behavior: (1) Wait for a new request on the server ZMQ socket (2) Create a [Server Operation](#) (3) Enqueue onto operation_queue (4) Goto step (1)

7.29.3.7 `std::thread zcm::Server::spawn ()`

Spawn a new thread for the server.

Returns

[Server](#) thread

7.29.3.8 `void zcm::Server::start ()`

Start the server thread.

7.29.4 Member Data Documentation

7.29.4.1 `zmq::context_t* zcm::Server::context` `[private]`

Pointer to the server ZMQ context.

7.29.4.2 `std::vector<std::string> zcm::Server::endpoints` [private]

Vector of connection endpoints.

7.29.4.3 `std::mutex zcm::Server::func_mutex` [private]

Mutex used when changing operation_function at runtime.

7.29.4.4 `std::string zcm::Server::name` [private]

Name of the server.

7.29.4.5 `std::function<std::string(const std::string&> zcm::Server::operation_function` [private]

Operation function bound to the server - [Component](#) method that handles received requests.

7.29.4.6 `Operation_Queue* zcm::Server::operation_queue_ptr` [private]

Pointer to the operation_queue.

7.29.4.7 `unsigned int zcm::Server::priority` [private]

Priority of the server.

7.29.4.8 `bool zcm::Server::ready` [private]

Boolean representing the state of the server to receive new requests.

7.29.4.9 `zmq::socket_t* zcm::Server::server_socket` [private]

Pointer to the server ZMQ socket.

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

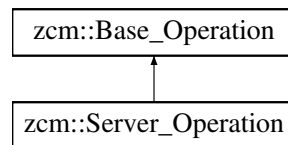
- [/home/pranav/Repositories/zcm/include/server.hpp](#)
- [/home/pranav/Repositories/zcm/src/server.cpp](#)

7.30 zcm::Server_Operation Class Reference

Server Operation class.

```
#include <operation_types.hpp>
```

Inheritance diagram for zcm::Server_Operation:



Public Member Functions

- [Server_Operation](#) (std::string [name](#), unsigned int [priority](#), std::function< std::string()> [operation_function](#), zmq::socket_t *[socket_ptr](#), bool *[recv_ready](#))
Construct a server operation.
- void [execute](#) ()
Server operation function.
- zmq::socket_t * [get_socket_ptr](#) ()
Get the ZMQ server socket pointer.
- void [set_ready](#) ()
Get the ZMQ server "ready" variable.
- std::string [get_name](#) ()
Return the operation name.
- unsigned int [get_priority](#) () const
Return the operation priority.

Private Attributes

- std::function< std::string()> [operation_function](#)
Server Operation Function.
- zmq::socket_t * [socket_ptr](#)
Pointer to the Server ZMQ socket.
- bool * [recv_ready](#)
Pointer to the Server "ready" variable.

7.30.1 Detailed Description

Server Operation class.

7.30.2 Constructor & Destructor Documentation

- 7.30.2.1 `zcm::Server_Operation::Server_Operation (std::string name, unsigned int priority, std::function< std::string()> operation_function, zmq::socket_t * socket_ptr, bool * recv_ready) [inline]`

Construct a server operation.

Parameters

in	<i>name</i>	Name of the operation
in	<i>priority</i>	Priority of the operation
in	<i>operation_function</i>	Server function
in	<i>socket_ptr</i>	Pointer to the Server ZMQ socket
in	<i>recv_ready</i>	Pointer to the Server ready variable

7.30.3 Member Function Documentation

7.30.3.1 `void zcm::Server_Operation::execute ()` [virtual]

[Server](#) operation function.

Reimplemented from [zcm::Base_Operation](#).

7.30.3.2 `std::string zcm::Base_Operation::get_name ()` [inherited]

Return the operation name.

Returns

Name of the operation

7.30.3.3 `unsigned int zcm::Base_Operation::get_priority () const` [inherited]

Return the operation priority.

Returns

Priority of the operation

7.30.3.4 `zmq::socket_t * zcm::Server_Operation::get_socket_ptr ()`

Get the ZMQ server socket pointer.

7.30.3.5 `void zcm::Server_Operation::set_ready ()`

Get the ZMQ server "ready" variable.

7.30.4 Member Data Documentation

7.30.4.1 `std::function<std::string()> zcm::Server_Operation::operation_function` [private]

[Server](#) Operation Function.

7.30.4.2 `bool* zcm::Server_Operation::recv_ready` [private]

Pointer to the [Server](#) "ready" variable.

7.30.4.3 `zmq::socket_t* zcm::Server_Operation::socket_ptr` [private]

Pointer to the [Server](#) ZMQ socket.

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

- [/home/pranav/Repositories/zcm/include/operation_types.hpp](#)
- [/home/pranav/Repositories/zcm/src/operation_types.cpp](#)

7.31 Json::StaticString Class Reference

Lightweight wrapper to tag static string.

```
#include <json.hpp>
```

Public Member Functions

- [StaticString](#) (const char *cstring)
- [operator const char *](#) () const
- const char * [c_str](#) () const

Private Attributes

- const char * [c_str_](#)

7.31.1 Detailed Description

Lightweight wrapper to tag static string.

[Value](#) constructor and `objectValue` member assignment takes advantage of the [StaticString](#) and avoid the cost of string duplication when storing the string or the member name.

Example of usage:

```
Json::Value aValue( StaticString("some text") );
Json::Value object;
static const StaticString code("code");
object[code] = 1234;
```

7.31.2 Constructor & Destructor Documentation

7.31.2.1 `Json::StaticString::StaticString (const char * czstring)` `[inline]`, `[explicit]`

7.31.3 Member Function Documentation

7.31.3.1 `const char* Json::StaticString::c_str () const` `[inline]`

7.31.3.2 `Json::StaticString::operator const char * () const` `[inline]`

7.31.4 Member Data Documentation

7.31.4.1 `const char* Json::StaticString::c_str_` `[private]`

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

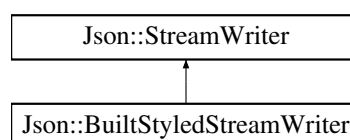
- </home/pranav/Repositories/zcm/include/json.hpp>

7.32 Json::StreamWriter Class Reference

Usage:

```
#include <json.hpp>
```

Inheritance diagram for Json::StreamWriter:



Classes

- class [Factory](#)
A simple abstract factory.

Public Member Functions

- [StreamWriter](#) ()
- virtual [~StreamWriter](#) ()
- virtual int [write](#) ([Value](#) const &root, [JSONCPP_OSTREAM](#) *sout)=0
Write [Value](#) into document as configured in sub-class.

Protected Attributes

- [JSONCPP_OSTREAM](#) * [sout_](#)

7.32.1 Detailed Description

Usage:

```
using namespace Json;
void writeToStdout(StreamWriter::Factory const& factory,
    Value const& value) {
    std::unique_ptr<StreamWriter> const writer(
        factory.newStreamWriter());
    writer->write(value, &std::cout);
    std::cout << std::endl; // add lf and flush
}
```

7.32.2 Constructor & Destructor Documentation

7.32.2.1 [Json::StreamWriter::StreamWriter \(\)](#)

7.32.2.2 [Json::StreamWriter::~~StreamWriter \(\)](#) [virtual]

7.32.3 Member Function Documentation

7.32.3.1 [virtual int Json::StreamWriter::write \(Value const & root, JSONCPP_OSTREAM * sout \)](#) [pure virtual]

Write [Value](#) into document as configured in sub-class.

Do not take ownership of sout, but maintain a reference during function.

Precondition

`sout != NULL`

Returns

zero on success (For now, we always return zero, so check the stream instead.)

Exceptions

<code>std::exception</code>	possibly, depending on configuration
-----------------------------	--------------------------------------

Implemented in [Json::BuiltStyledStreamWriter](#).

7.32.4 Member Data Documentation

7.32.4.1 JSONCPP_OSTREAM* Json::StreamWriter::sout_ [protected]

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

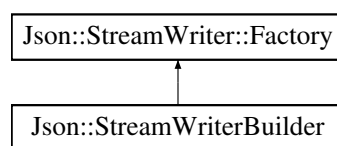
- /home/pranav/Repositories/zcm/include/json.hpp
- /home/pranav/Repositories/zcm/src/json.cpp

7.33 Json::StreamWriterBuilder Class Reference

Build a [StreamWriter](#) implementation.

```
#include <json.hpp>
```

Inheritance diagram for Json::StreamWriterBuilder:



Public Member Functions

- [StreamWriterBuilder](#) ()
- [~StreamWriterBuilder](#) () [JSONCPP_OVERRIDE](#)
- [StreamWriter](#) * [newStreamWriter](#) () const [JSONCPP_OVERRIDE](#)
- bool [validate](#) (Json::Value *invalid) const
- [Value](#) & [operator\[\]](#) ([JSONCPP_STRING](#) key)

A simple way to update a specific setting.

Static Public Member Functions

- static void [setDefaults](#) (Json::Value *settings)
- Called by ctor, but you can use this to reset settings_.*

Public Attributes

- [Json::Value](#) [settings_](#)
- Configuration of this builder.*

7.33.1 Detailed Description

Build a [StreamWriter](#) implementation.

Usage:

```
using namespace Json;
Value value = ...;
StreamWriterBuilder builder;
builder["commentStyle"] = "None";
builder["indentation"] = "  "; // or whatever you like
std::unique_ptr<Json::StreamWriter> writer(
    builder.newStreamWriter());
writer->write(value, &std::cout);
std::cout << std::endl; // add lf and flush
```

7.33.2 Constructor & Destructor Documentation

7.33.2.1 `Json::StreamWriterBuilder::StreamWriterBuilder ()`

7.33.2.2 `Json::StreamWriterBuilder::~~StreamWriterBuilder ()`

7.33.3 Member Function Documentation

7.33.3.1 `StreamWriter * Json::StreamWriterBuilder::newStreamWriter () const` `[virtual]`

Exceptions

<code>std::exception</code>	if something goes wrong (e.g. invalid settings)
-----------------------------	---

Implements [Json::StreamWriter::Factory](#).

7.33.3.2 `Value & Json::StreamWriterBuilder::operator[] (JSONCPP_STRING key)`

A simple way to update a specific setting.

7.33.3.3 `void Json::StreamWriterBuilder::setDefaults (Json::Value * settings)` `[static]`

Called by ctor, but you can use this to reset settings_.

Precondition

'settings' != NULL (but Json::null is fine)

Remarks

Defaults:

[StreamWriterBuilderDefaults]

[StreamWriterBuilderDefaults]

7.33.3.4 `bool Json::StreamWriterBuilder::validate (Json::Value * invalid) const`

Returns

true if 'settings' are legal and consistent; otherwise, indicate bad settings via 'invalid'.

7.33.4 Member Data Documentation

7.33.4.1 `Json::Value Json::StreamWriterBuilder::settings_`

Configuration of this builder.

Available settings (case-sensitive):

- "commentStyle": "None" or "All"
- "indentation": "<anything>"
- "enableYAMLCompatibility": false or true
 - slightly change the whitespace around colons
- "dropNullPlaceholders": false or true
 - Drop the "null" string from the writer's output for nullValues. Strictly speaking, this is not valid JSON. But when the output is being fed to a browser's Javascript, it makes for smaller output and the browser can handle the output just fine.
- "useSpecialFloats": false or true
 - If true, outputs non-finite floating point values in the following way: NaN values as "NaN", positive infinity as "Infinity", and negative infinity as "-Infinity".

You can examine 'settings_' yourself to see the defaults. You can also write and read them just like any JSON [Value](#).

See also

[setDefault\(\)](#)

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.34 `Json::Value::CZString::StringStorage` Struct Reference

Public Attributes

- unsigned [policy_](#): 2
- unsigned [length_](#): 30

7.34.1 Member Data Documentation

7.34.1.1 `unsigned Json::Value::CZString::StringStorage::length_`

7.34.1.2 `unsigned Json::Value::CZString::StringStorage::policy_`

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)

7.35 Json::Reader::StructuredError Struct Reference

An error tagged with where in the JSON text it was encountered.

```
#include <json.hpp>
```

Public Attributes

- [ptrdiff_t offset_start](#)
- [ptrdiff_t offset_limit](#)
- [JSONCPP_STRING](#) message

7.35.1 Detailed Description

An error tagged with where in the JSON text it was encountered.

The offsets give the [start, limit) range of bytes within the text. Note that this is bytes, not codepoints.

7.35.2 Member Data Documentation

7.35.2.1 [JSONCPP_STRING](#) [Json::Reader::StructuredError::message](#)

7.35.2.2 [ptrdiff_t](#) [Json::Reader::StructuredError::offset_limit](#)

7.35.2.3 [ptrdiff_t](#) [Json::Reader::StructuredError::offset_start](#)

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)

7.36 Json::OurReader::StructuredError Struct Reference

Public Attributes

- [ptrdiff_t offset_start](#)
- [ptrdiff_t offset_limit](#)
- [JSONCPP_STRING](#) message

7.36.1 Member Data Documentation

7.36.1.1 [JSONCPP_STRING](#) [Json::OurReader::StructuredError::message](#)

7.36.1.2 [ptrdiff_t](#) [Json::OurReader::StructuredError::offset_limit](#)

7.36.1.3 [ptrdiff_t](#) [Json::OurReader::StructuredError::offset_start](#)

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

- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.37 Json::StyledStreamWriter Class Reference

Writes a [Value](#) in [JSON](#) format in a human friendly way, to a stream rather than to a string.

```
#include <json.hpp>
```

Public Member Functions

- [StyledStreamWriter](#) ([JSONCPP_STRING](#) indentation="[t](#)")
- [~StyledStreamWriter](#) ()
- void [write](#) ([JSONCPP_OSTREAM](#) &out, const [Value](#) &root)
Serialize a [Value](#) in [JSON](#) format.

Private Types

- typedef std::vector< [JSONCPP_STRING](#) > [ChildValues](#)

Private Member Functions

- void [writeValue](#) (const [Value](#) &value)
- void [writeArrayValue](#) (const [Value](#) &value)
- bool [isMultilineArray](#) (const [Value](#) &value)
- void [pushValue](#) (const [JSONCPP_STRING](#) &value)
- void [writeIndent](#) ()
- void [writeWithIndent](#) (const [JSONCPP_STRING](#) &value)
- void [indent](#) ()
- void [unindent](#) ()
- void [writeCommentBeforeValue](#) (const [Value](#) &root)
- void [writeCommentAfterValueOnSameLine](#) (const [Value](#) &root)
- bool [hasCommentForValue](#) (const [Value](#) &value)

Static Private Member Functions

- static [JSONCPP_STRING](#) [normalizeEOL](#) (const [JSONCPP_STRING](#) &text)

Private Attributes

- [ChildValues](#) [childValues_](#)
- [JSONCPP_OSTREAM](#) * [document_](#)
- [JSONCPP_STRING](#) [indentString_](#)
- unsigned int [rightMargin_](#)
- [JSONCPP_STRING](#) [indentation_](#)
- bool [addChildValues_](#): 1
- bool [indented_](#): 1

7.37.1 Detailed Description

Writes a [Value](#) in [JSON](#) format in a human friendly way, to a stream rather than to a string.

The rules for line break and indent are as follow:

- Object value:
 - if empty then print {} without indent and line break
 - if not empty the print '{', line break & indent, print one value per line and then unindent and line break and print '}'.
- Array value:
 - if empty then print [] without indent and line break
 - if the array contains no object value, empty array or some other value types, and all the values fit on one lines, then print the array on a single line.
 - otherwise, if the values do not fit on one line, or the array contains object or non empty array, then print one value per line.

If the [Value](#) have comments then they are outputed according to their [CommentPlacement](#).

Parameters

<i>indentation</i>	Each level will be indented by this amount extra.
--------------------	---

See also

[Reader](#), [Value](#), [Value::setComment\(\)](#)

Deprecated Use [StreamWriterBuilder](#).

7.37.2 Member Typedef Documentation

7.37.2.1 `typedef std::vector<JSONCPP_STRING> Json::StyledStreamWriter::ChildValues` `[private]`

7.37.3 Constructor & Destructor Documentation

7.37.3.1 `Json::StyledStreamWriter::StyledStreamWriter (JSONCPP_STRING indentation = "\t")`

7.37.3.2 `Json::StyledStreamWriter::~~StyledStreamWriter ()` `[inline]`

7.37.4 Member Function Documentation

7.37.4.1 `bool Json::StyledStreamWriter::hasCommentForValue (const Value & value)` `[private]`

7.37.4.2 `void Json::StyledStreamWriter::indent ()` `[private]`

7.37.4.3 `bool Json::StyledStreamWriter::isMultilineArray (const Value & value) [private]`

7.37.4.4 `static JSONCPP_STRING Json::StyledStreamWriter::normalizeEOL (const JSONCPP_STRING & text) [static], [private]`

7.37.4.5 `void Json::StyledStreamWriter::pushValue (const JSONCPP_STRING & value) [private]`

7.37.4.6 `void Json::StyledStreamWriter::unindent () [private]`

7.37.4.7 `void Json::StyledStreamWriter::write (JSONCPP_OSTREAM & out, const Value & root)`

Serialize a [Value](#) in [JSON](#) format.

Parameters

<i>out</i>	Stream to write to. (Can be ostream, e.g.)
<i>root</i>	Value to serialize.

Note

There is no point in deriving from [Writer](#), since `write()` should not return a value.

7.37.4.8 `void Json::StyledStreamWriter::writeArrayValue (const Value & value) [private]`

7.37.4.9 `void Json::StyledStreamWriter::writeCommentAfterValueOnSameLine (const Value & root) [private]`

7.37.4.10 `void Json::StyledStreamWriter::writeCommentBeforeValue (const Value & root) [private]`

7.37.4.11 `void Json::StyledStreamWriter::writeIndent () [private]`

7.37.4.12 `void Json::StyledStreamWriter::writeValue (const Value & value) [private]`

7.37.4.13 `void Json::StyledStreamWriter::writeWithIndent (const JSONCPP_STRING & value) [private]`

7.37.5 Member Data Documentation

7.37.5.1 `bool Json::StyledStreamWriter::addChildValues_ [private]`

7.37.5.2 `ChildValues Json::StyledStreamWriter::childValues_ [private]`

7.37.5.3 `JSONCPP_OSTREAM* Json::StyledStreamWriter::document_ [private]`

7.37.5.4 `JSONCPP_STRING Json::StyledStreamWriter::indentation_ [private]`

7.37.5.5 `bool Json::StyledStreamWriter::indented_ [private]`

7.37.5.6 `JSONCPP_STRING Json::StyledStreamWriter::indentString_ [private]`

7.37.5.7 `unsigned int Json::StyledStreamWriter::rightMargin_ [private]`

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

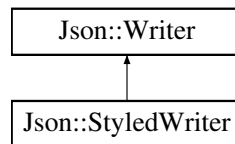
- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.38 Json::StyledWriter Class Reference

Writes a [Value](#) in [JSON](#) format in a human friendly way.

```
#include <json.hpp>
```

Inheritance diagram for Json::StyledWriter:



Public Member Functions

- [StyledWriter](#) ()
- [~StyledWriter](#) () [JSONCPP_OVERRIDE](#)
- [JSONCPP_STRING write](#) (const [Value](#) &root) [JSONCPP_OVERRIDE](#)
Serialize a [Value](#) in [JSON](#) format.

Private Types

- typedef std::vector< [JSONCPP_STRING](#) > [ChildValues](#)

Private Member Functions

- void [writeValue](#) (const [Value](#) &value)
- void [writeArrayValue](#) (const [Value](#) &value)
- bool [isMultilineArray](#) (const [Value](#) &value)
- void [pushValue](#) (const [JSONCPP_STRING](#) &value)
- void [writeIndent](#) ()
- void [writeWithIndent](#) (const [JSONCPP_STRING](#) &value)
- void [indent](#) ()
- void [unindent](#) ()
- void [writeCommentBeforeValue](#) (const [Value](#) &root)
- void [writeCommentAfterValueOnSameLine](#) (const [Value](#) &root)
- bool [hasCommentForValue](#) (const [Value](#) &value)

Static Private Member Functions

- static [JSONCPP_STRING normalizeEOL](#) (const [JSONCPP_STRING](#) &text)

Private Attributes

- [ChildValues](#) [childValues_](#)
- [JSONCPP_STRING](#) [document_](#)
- [JSONCPP_STRING](#) [indentString_](#)
- unsigned int [rightMargin_](#)
- unsigned int [indentSize_](#)
- bool [addChildValues_](#)

7.38.1 Detailed Description

Writes a [Value](#) in [JSON](#) format in a human friendly way.

The rules for line break and indent are as follow:

- Object value:
 - if empty then print {} without indent and line break
 - if not empty the print '{', line break & indent, print one value per line and then unindent and line break and print '}'.
- Array value:
 - if empty then print [] without indent and line break
 - if the array contains no object value, empty array or some other value types, and all the values fit on one lines, then print the array on a single line.
 - otherwise, if the values do not fit on one line, or the array contains object or non empty array, then print one value per line.

If the [Value](#) have comments then they are outputed according to their [CommentPlacement](#).

See also

[Reader](#), [Value](#), [Value::setComment\(\)](#)

Deprecated Use [StreamWriterBuilder](#).

7.38.2 Member Typedef Documentation

7.38.2.1 `typedef std::vector<JSONCPP_STRING> Json::StyledWriter::ChildValues` `[private]`

7.38.3 Constructor & Destructor Documentation

7.38.3.1 `Json::StyledWriter::StyledWriter ()`

7.38.3.2 `Json::StyledWriter::~~StyledWriter ()` `[inline]`

7.38.4 Member Function Documentation

7.38.4.1 `bool Json::StyledWriter::hasCommentForValue (const Value & value)` `[private]`

7.38.4.2 `void Json::StyledWriter::indent ()` `[private]`

7.38.4.3 `bool Json::StyledWriter::isMultilineArray (const Value & value)` `[private]`

7.38.4.4 `static JSONCPP_STRING Json::StyledWriter::normalizeEOL (const JSONCPP_STRING & text)`
`[static], [private]`

7.38.4.5 `void Json::StyledWriter::pushValue (const JSONCPP_STRING & value)` `[private]`

7.38.4.6 `void Json::StyledWriter::unindent ()` `[private]`

7.38.4.7 `JSONCPP_STRING Json::StyledWriter::write (const Value & root)` `[virtual]`

Serialize a [Value](#) in [JSON](#) format.

Parameters

<i>root</i>	Value to serialize.
-------------	---------------------

Returns

String containing the JSON document that represents the root value.

Implements [Json::Writer](#).

7.38.4.8 void `Json::StyledWriter::writeArrayValue (const Value & value)` [private]

7.38.4.9 void `Json::StyledWriter::writeCommentAfterValueOnSameLine (const Value & root)` [private]

7.38.4.10 void `Json::StyledWriter::writeCommentBeforeValue (const Value & root)` [private]

7.38.4.11 void `Json::StyledWriter::writeIndent ()` [private]

7.38.4.12 void `Json::StyledWriter::writeValue (const Value & value)` [private]

7.38.4.13 void `Json::StyledWriter::writeWithIndent (const JSONCPP_STRING & value)` [private]

7.38.5 Member Data Documentation

7.38.5.1 bool `Json::StyledWriter::addChildValues_` [private]

7.38.5.2 ChildValues `Json::StyledWriter::childValues_` [private]

7.38.5.3 JSONCPP_STRING `Json::StyledWriter::document_` [private]

7.38.5.4 unsigned int `Json::StyledWriter::indentSize_` [private]

7.38.5.5 JSONCPP_STRING `Json::StyledWriter::indentString_` [private]

7.38.5.6 unsigned int `Json::StyledWriter::rightMargin_` [private]

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.39 zcm::Subscriber Class Reference

[Subscriber](#) class.

```
#include <subscriber.hpp>
```

Public Member Functions

- [Subscriber](#) (std::string [name](#), unsigned int [priority](#), std::string [filter](#), std::function< void(const std::string &)> [operation_function](#), [Operation_Queue](#) *[operation_queue_ptr](#))
Construct a subscriber object.
- [Subscriber](#) (std::string [name](#), unsigned int [priority](#), std::string [filter](#), std::vector< std::string > [endpoints](#), std::function< void(const std::string &)> [operation_function](#), [Operation_Queue](#) *[operation_queue_ptr](#))
Construct a subscriber object with known endpoints.
- [~Subscriber](#) ()
Close the subscriber socket and destroy the ZMQ context.
- void [connect](#) (std::vector< std::string > [new_endpoints](#))
Connect to a new set of endpoints param[in] new_endpoints A new vector of endpoints to connect to.
- std::string [get_name](#) ()
Get the name of the subscriber.
- unsigned int [get_priority](#) ()
Get the priority of the subscriber.
- void [add_connection](#) (std::string [new_connection](#))
Add a new connection to the subscriber.
- void [recv](#) ()
Thread function of the subscriber Behavior: (1) Wait for a new message on the subscriber ZMQ socket (2) Create a Subscriber Operation (3) Enqueue onto operation_queue (4) Goto step (1)
- void [rebind_operation_function](#) (std::function< void(const std::string &)> [new_operation_function](#))
Rebind the subscriber operation function.
- std::thread [spawn](#) ()
Spawn a new thread for the subscriber.
- void [start](#) ()
Start the subscriber thread.

Private Attributes

- std::string [name](#)
Name of the subscriber.
- unsigned int [priority](#)
Priority of the subscriber.
- std::string [filter](#)
Reception filter enforced on all received messages.
- std::vector< std::string > [endpoints](#)
Vector of connection endpoints.
- std::function< void(const std::string &)> [operation_function](#)
Operation function bound to the subscriber - [Component](#) method that handles received message.
- [Operation_Queue](#) * [operation_queue_ptr](#)
Pointer to the operation queue.
- zmq::context_t * [context](#)
Pointer to the subscriber ZMQ context.
- zmq::socket_t * [subscriber_socket](#)
Pointer to the subscriber ZMQ socket.
- std::mutex [func_mutex](#)
Mutex used to change operation_function at runtime.

7.39.1 Detailed Description

[Subscriber](#) class.

7.39.2 Constructor & Destructor Documentation

7.39.2.1 `zcm::Subscriber::Subscriber (std::string name, unsigned int priority, std::string filter, std::function< void(const std::string &)> operation_function, Operation_Queue * operation_queue_ptr)` `[inline]`

Construct a subscriber object.

Parameters

in	<i>name</i>	Subscriber name
in	<i>priority</i>	Priority of the subscriber
in	<i>filter</i>	ZMQ filter for the subscriber
in	<i>operation_function</i>	Operation function of the subscriber
in	<i>operation_queue_ptr</i>	Pointer to the operation queue

7.39.2.2 `zcm::Subscriber::Subscriber (std::string name, unsigned int priority, std::string filter, std::vector< std::string > endpoints, std::function< void(const std::string &)> operation_function, Operation_Queue * operation_queue_ptr)`

Construct a subscriber object with known endpoints.

Parameters

in	<i>name</i>	Subscriber name
in	<i>priority</i>	Priority of the subscriber
in	<i>filter</i>	ZMQ filter for the subscriber
in	<i>endpoints</i>	A vector of endpoints to connect to
in	<i>operation_function</i>	Operation function of the subscriber
in	<i>operation_queue_ptr</i>	Pointer to the operation queue

7.39.2.3 `zcm::Subscriber::~~Subscriber ()`

Close the subscriber socket and destroy the ZMQ context.

7.39.3 Member Function Documentation

7.39.3.1 `void zcm::Subscriber::add_connection (std::string new_connection)`

Add a new connection to the subscriber.

Parameters

in	<i>new_connection</i>	New connection address to connect to
----	-----------------------	--------------------------------------

7.39.3.2 void zcm::Subscriber::connect (std::vector< std::string > *new_endpoints*)

Connect to a new set of endpoints param[in] *new_endpoints* A new vector of endpoints to connect to.

7.39.3.3 std::string zcm::Subscriber::get_name ()

Get the name of the subscriber.

7.39.3.4 unsigned int zcm::Subscriber::get_priority ()

Get the priority of the subscriber.

7.39.3.5 void zcm::Subscriber::rebind_operation_function (std::function< void(const std::string &)> *new_operation_function*)

Rebind the subscriber operation function.

Parameters

in	<i>new_operation_function</i>	New subscriber function to be handled upon recv()
----	-------------------------------	---

7.39.3.6 void zcm::Subscriber::recv ()

Thread function of the subscriber Behavior: (1) Wait for a new message on the subscriber ZMQ socket (2) Create a Subscriber Operation (3) Enqueue onto operation_queue (4) Goto step (1)

7.39.3.7 std::thread zcm::Subscriber::spawn ()

Spawn a new thread for the subscriber.

Returns

[Subscriber](#) thread

7.39.3.8 void zcm::Subscriber::start ()

Start the subscriber thread.

7.39.4 Member Data Documentation

7.39.4.1 `zmq::context_t* zcm::Subscriber::context` [private]

Pointer to the subscriber ZMQ context.

7.39.4.2 `std::vector<std::string> zcm::Subscriber::endpoints` [private]

Vector of connection endpoints.

7.39.4.3 `std::string zcm::Subscriber::filter` [private]

Reception filter enforced on all received messages.

7.39.4.4 `std::mutex zcm::Subscriber::func_mutex` [private]

Mutex used to change operation_function at runtime.

7.39.4.5 `std::string zcm::Subscriber::name` [private]

Name of the subscriber.

7.39.4.6 `std::function<void(const std::string&)> zcm::Subscriber::operation_function` [private]

Operation function bound to the subscriber - [Component](#) method that handles received message.

7.39.4.7 `Operation_Queue* zcm::Subscriber::operation_queue_ptr` [private]

Pointer to the operation queue.

7.39.4.8 `unsigned int zcm::Subscriber::priority` [private]

Priority of the subscriber.

7.39.4.9 `zmq::socket_t* zcm::Subscriber::subscriber_socket` [private]

Pointer to the subscriber ZMQ socket.

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

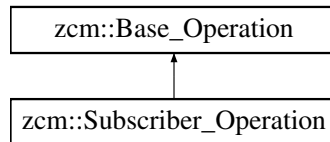
- [/home/pranav/Repositories/zcm/include/subscriber.hpp](#)
- [/home/pranav/Repositories/zcm/src/subscriber.cpp](#)

7.40 zcm::Subscriber_Operation Class Reference

[Subscriber](#) Operation class.

```
#include <operation_types.hpp>
```

Inheritance diagram for zcm::Subscriber_Operation:



Public Member Functions

- [Subscriber_Operation](#) (std::string [name](#), unsigned int [priority](#), std::function< void()> [operation_function](#))
Construct a subscriber operation.
- void [execute](#) ()
[Subscriber](#) operation function.
- std::string [get_name](#) ()
Return the operation name.
- unsigned int [get_priority](#) () const
Return the operation priority.

Private Attributes

- std::function< void()> [operation_function](#)
[Subscriber](#) Operation Function.

7.40.1 Detailed Description

[Subscriber](#) Operation class.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 `zcm::Subscriber_Operation::Subscriber_Operation (std::string name, unsigned int priority, std::function< void()> operation_function)` `[inline]`

Construct a subscriber operation.

Parameters

in	<i>name</i>	Name of the operation
in	<i>priority</i>	Priority of the operation
in	<i>operation_function</i>	Subscriber function

7.40.3 Member Function Documentation

7.40.3.1 `void zcm::Subscriber_Operation::execute () [virtual]`

[Subscriber](#) operation function.

Reimplemented from [zcm::Base_Operation](#).

7.40.3.2 `std::string zcm::Base_Operation::get_name () [inherited]`

Return the operation name.

Returns

Name of the operation

7.40.3.3 `unsigned int zcm::Base_Operation::get_priority () const [inherited]`

Return the operation priority.

Returns

Priority of the operation

7.40.4 Member Data Documentation

7.40.4.1 `std::function<void()> zcm::Subscriber_Operation::operation_function [private]`

[Subscriber](#) Operation Function.

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

- [/home/pranav/Repositories/zcm/include/operation_types.hpp](#)
- [/home/pranav/Repositories/zcm/src/operation_types.cpp](#)

7.41 zcm::Timer Class Reference

[Timer](#) class.

```
#include <timer.hpp>
```

Public Member Functions

- [Timer](#) (std::string [name](#), unsigned int [priority](#), long long [period](#), std::function< void()> [operation_function](#), [Operation_Queue](#) *[operation_queue_ptr](#))
Construct a timer.
- void [operation](#) ()
[Timer](#) thread function Behavior: (1) Wait for timer expiry (2) Create a [Timer_Operation](#) (3) Enqueue onto operation←→_queue (4) Goto step (1)
- std::string [get_name](#) ()
Get the timer name.
- unsigned int [get_priority](#) ()
Get the timer priority.
- void [change_period](#) (long long new_period)
Change the timer period.
- void [rebind_operation_function](#) (std::function< void()> new_operation_function)
Rebind the timer operation function.
- std::thread [spawn](#) ()
Spawn a new thread for the timer.
- void [start](#) ()
Start the timer thread.

Private Attributes

- std::string [name](#)
Name of the timer.
- unsigned int [priority](#)
Priority of the timer.
- std::chrono::duration< long long, std::ratio< 1, 1000000000 > > [period](#)
Period of the timer.
- std::function< void()> [operation_function](#)
Operation function bound to the timer.
- [Operation_Queue](#) * [operation_queue_ptr](#)
Pointer to the operation queue.
- std::mutex [period_mutex](#)
Mutex used to change the timer period at runtime.
- std::mutex [func_mutex](#)
Mutex used to change the operation_function at runtime.

7.41.1 Detailed Description

[Timer](#) class.

7.41.2 Constructor & Destructor Documentation

- 7.41.2.1 `zcm::Timer::Timer (std::string name, unsigned int priority, long long period, std::function< void()> operation_function, Operation_Queue * operation_queue_ptr)`

Construct a timer.

Parameters

in	<i>name</i>	Name of the timer
in	<i>priority</i>	Priority of the timer
in	<i>period</i>	Period of the timer in nanoseconds
in	<i>operation_function</i>	Operation to which the timer is bound
in	<i>operation_queue_ptr</i>	Pointer to the operation_queue

7.41.3 Member Function Documentation

7.41.3.1 void zcm::Timer::change_period (long long *new_period*)

Change the timer period.

Parameters

in	<i>new_period</i>	New timer period in nanoseconds
----	-------------------	---------------------------------

7.41.3.2 std::string zcm::Timer::get_name ()

Get the timer name.

Returns

[Timer](#) name

7.41.3.3 unsigned int zcm::Timer::get_priority ()

Get the timer priority.

Returns

[Timer](#) priority

7.41.3.4 void zcm::Timer::operation ()

[Timer](#) thread function Behavior: (1) Wait for timer expiry (2) Create a [Timer_Operation](#) (3) Enqueue onto operation_queue (4) Goto step (1)

7.41.3.5 void zcm::Timer::rebind_operation_function (std::function< void()> *new_operation_function*)

Rebind the timer operation function.

Parameters

in	<i>new_operation_function</i>	New timer function to be handled upon expiry
----	-------------------------------	--

7.41.3.6 `std::thread zcm::Timer::spawn ()`

Spawn a new thread for the timer.

Returns

[Timer](#) thread

7.41.3.7 `void zcm::Timer::start ()`

Start the timer thread.

7.41.4 Member Data Documentation

7.41.4.1 `std::mutex zcm::Timer::func_mutex` [private]

Mutex used to change the operation_function at runtime.

7.41.4.2 `std::string zcm::Timer::name` [private]

Name of the timer.

7.41.4.3 `std::function<void()> zcm::Timer::operation_function` [private]

Operation function bound to the timer.

7.41.4.4 `Operation_Queue* zcm::Timer::operation_queue_ptr` [private]

Pointer to the operation queue.

7.41.4.5 `std::chrono::duration<long long, std::ratio<1, 1000000000> > zcm::Timer::period` [private]

Period of the timer.

7.41.4.6 `std::mutex zcm::Timer::period_mutex` [private]

Mutex used to change the timer period at runtime.

7.41.4.7 unsigned int zcm::Timer::priority [private]

Priority of the timer.

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

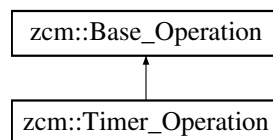
- /home/pranav/Repositories/zcm/include/timer.hpp
- /home/pranav/Repositories/zcm/src/timer.cpp

7.42 zcm::Timer_Operation Class Reference

Timer Operation class.

```
#include <operation_types.hpp>
```

Inheritance diagram for zcm::Timer_Operation:



Public Member Functions

- [Timer_Operation](#) (std::string [name](#), unsigned int [priority](#), std::function< void()> [operation_function](#))
Construct a timer operation.
- void [execute](#) ()
Timer operation function.
- std::string [get_name](#) ()
Return the operation name.
- unsigned int [get_priority](#) () const
Return the operation priority.

Private Attributes

- std::function< void()> [operation_function](#)
Timer operation function.

7.42.1 Detailed Description

Timer Operation class.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 zcm::Timer_Operation::Timer_Operation (std::string *name*, unsigned int *priority*, std::function< void()> *operation_function*) [inline]

Construct a timer operation.

Parameters

in	<i>name</i>	Name of the operation
in	<i>priority</i>	Priority of the operation
in	<i>operation_function</i>	Timer function

7.42.3 Member Function Documentation

7.42.3.1 `void zcm::Timer_Operation::execute ()` `[virtual]`

[Timer](#) operation function.

Reimplemented from [zcm::Base_Operation](#).

7.42.3.2 `std::string zcm::Base_Operation::get_name ()` `[inherited]`

Return the operation name.

Returns

Name of the operation

7.42.3.3 `unsigned int zcm::Base_Operation::get_priority () const` `[inherited]`

Return the operation priority.

Returns

Priority of the operation

7.42.4 Member Data Documentation

7.42.4.1 `std::function<void()> zcm::Timer_Operation::operation_function` `[private]`

[Timer](#) operation function.

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

- [/home/pranav/Repositories/zcm/include/operation_types.hpp](#)
- [/home/pranav/Repositories/zcm/src/operation_types.cpp](#)

7.43 Json::OurReader::Token Class Reference

Public Attributes

- [TokenType type_](#)
- [Location start_](#)
- [Location end_](#)

7.43.1 Member Data Documentation

7.43.1.1 **Location** Json::OurReader::Token::end_

7.43.1.2 **Location** Json::OurReader::Token::start_

7.43.1.3 **TokenType** Json::OurReader::Token::type_

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

- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.44 Json::Reader::Token Class Reference

Public Attributes

- [TokenType type_](#)
- [Location start_](#)
- [Location end_](#)

7.44.1 Member Data Documentation

7.44.1.1 **Location** Json::Reader::Token::end_

7.44.1.2 **Location** Json::Reader::Token::start_

7.44.1.3 **TokenType** Json::Reader::Token::type_

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)

7.45 Json::Value Class Reference

Represents a **JSON** value.

```
#include <json.hpp>
```

Classes

- struct [CommentInfo](#)
- class [CZString](#)
- union [ValueHolder](#)

Public Types

- typedef std::vector< [JSONCPP_STRING](#) > [Members](#)
- typedef [ValueIterator](#) iterator
- typedef [ValueConstIterator](#) const_iterator
- typedef [Json::UInt](#) UInt
- typedef [Json::Int](#) Int
- typedef [Json::UInt64](#) UInt64
- typedef [Json::Int64](#) Int64
- typedef [Json::LargestInt](#) LargestInt
- typedef [Json::LargestUInt](#) LargestUInt
- typedef [Json::ArrayIndex](#) ArrayIndex
- typedef std::map< [CZString](#), [Value](#) > [ObjectValues](#)

Public Member Functions

- [Value](#) ([ValueType](#) type=nullValue)
Create a default [Value](#) of the given type.
- [Value](#) ([Int](#) value)
- [Value](#) ([UInt](#) value)
- [Value](#) ([Int64](#) value)
- [Value](#) ([UInt64](#) value)
- [Value](#) (double value)
- [Value](#) (const char *value)
Copy til first 0. (NULL causes to seg-fault.)
- [Value](#) (const char *begin, const char *end)
Copy all, incl zeroes.
- [Value](#) (const [StaticString](#) &value)
Constructs a value from a static string.
- [Value](#) (const [JSONCPP_STRING](#) &value)
Copy data() til size(). Embedded zeroes too.
- [Value](#) (bool value)
- [Value](#) (const [Value](#) &other)
Deep copy.
- [~Value](#) ()
- [Value](#) & operator= ([Value](#) other)
Deep copy, then swap(other).
- void swap ([Value](#) &other)
Swap everything.
- void swapPayload ([Value](#) &other)
Swap values but leave comments and source offsets in place.
- [ValueType](#) type () const
- bool operator< (const [Value](#) &other) const
Compare payload only, not comments etc.
- bool operator<= (const [Value](#) &other) const

- bool `operator>=` (const `Value` &other) const
- bool `operator>` (const `Value` &other) const
- bool `operator==` (const `Value` &other) const
- bool `operator!=` (const `Value` &other) const
- int `compare` (const `Value` &other) const
- const char * `asCString` () const
Embedded zeroes could cause you trouble!
- `JSONCPP_STRING` `asString` () const
Embedded zeroes are possible.
- bool `getString` (char const **`begin`, char const **`end`) const
Get raw char of string-value.*
- `Int` `asInt` () const
- `UInt` `asUInt` () const
- `Int64` `asInt64` () const
- `UInt64` `asUInt64` () const
- `LargestInt` `asLargestInt` () const
- `LargestUInt` `asLargestUInt` () const
- float `asFloat` () const
- double `asDouble` () const
- bool `asBool` () const
- bool `isNull` () const
- bool `isBool` () const
- bool `isInt` () const
- bool `isInt64` () const
- bool `isUInt` () const
- bool `isUInt64` () const
- bool `isIntegral` () const
- bool `isDouble` () const
- bool `isNumeric` () const
- bool `isString` () const
- bool `isArray` () const
- bool `isObject` () const
- bool `isConvertibleTo` (`ValueType` other) const
- `ArrayIndex` `size` () const
Number of values in array or object.
- bool `empty` () const
Return true if empty array, empty object, or null; otherwise, false.
- bool `operator!` () const
Return isNull()
- void `clear` ()
Remove all object members and array elements.
- void `resize` (`ArrayIndex` size)
Resize the array to size elements.
- `Value` & `operator[]` (`ArrayIndex` index)
Access an array element (zero based index).
- `Value` & `operator[]` (int index)
Access an array element (zero based index).
- const `Value` & `operator[]` (`ArrayIndex` index) const
Access an array element (zero based index) (You may need to say 'value[0u]' to get your compiler to distinguish this from the operator[] which takes a string.)
- const `Value` & `operator[]` (int index) const
Access an array element (zero based index) (You may need to say 'value[0u]' to get your compiler to distinguish this from the operator[] which takes a string.)

- **Value** **get** (**ArrayIndex** index, const **Value** &defaultValue) const
If the array contains at least index+1 elements, returns the element value, otherwise returns defaultValue.
- bool **isValidIndex** (**ArrayIndex** index) const
Return true if index < size().
- **Value** & **append** (const **Value** &value)
Append value to array at the end.
- **Value** & **operator[]** (const char *key)
Access an object value by name, create a null member if it does not exist.
- const **Value** & **operator[]** (const char *key) const
Access an object value by name, returns null if there is no member with that name.
- **Value** & **operator[]** (const **JSONCPP_STRING** &key)
Access an object value by name, create a null member if it does not exist.
- const **Value** & **operator[]** (const **JSONCPP_STRING** &key) const
Access an object value by name, returns null if there is no member with that name.
- **Value** & **operator[]** (const **StaticString** &key)
Access an object value by name, create a null member if it does not exist.
- **Value** **get** (const char *key, const **Value** &defaultValue) const
Return the member named key if it exist, defaultValue otherwise.
- **Value** **get** (const char *begin, const char *end, const **Value** &defaultValue) const
Return the member named key if it exist, defaultValue otherwise.
- **Value** **get** (const **JSONCPP_STRING** &key, const **Value** &defaultValue) const
Return the member named key if it exist, defaultValue otherwise.
- **Value** const * **find** (char const *begin, char const *end) const
Most general and efficient version of isMember()const, get()const, and operator[]const.
- **Value** const * **demand** (char const *begin, char const *end)
Most general and efficient version of object-mutators.
- **Value** **removeMember** (const char *key)
Remove and return the named member.
- **Value** **removeMember** (const **JSONCPP_STRING** &key)
Same as removeMember(const char)*
- bool **removeMember** (const char *key, **Value** *removed)
Same as removeMember(const char begin, const char* end, Value* removed), but 'key' is null-terminated.*
- bool **removeMember** (**JSONCPP_STRING** const &key, **Value** *removed)
Remove the named map member.
- bool **removeMember** (const char *begin, const char *end, **Value** *removed)
Same as removeMember(JSONCPP_STRING const& key, Value removed)*
- bool **removeIndex** (**ArrayIndex** i, **Value** *removed)
Remove the indexed array element.
- bool **isMember** (const char *key) const
Return true if the object has a member named key.
- bool **isMember** (const **JSONCPP_STRING** &key) const
Return true if the object has a member named key.
- bool **isMember** (const char *begin, const char *end) const
Same as isMember(JSONCPP_STRING const& key)const.
- **Members** **getMemberNames** () const
Return a list of the member names.
- void **setComment** (const char *comment, **CommentPlacement** placement)
- void **setComment** (const char *comment, size_t len, **CommentPlacement** placement)
Comments must be //... or / ... */.*
- void **setComment** (const **JSONCPP_STRING** &comment, **CommentPlacement** placement)
Comments must be //... or / ... */.*

- bool `hasComment` (`CommentPlacement` placement) const
- `JSONCPP_STRING` `getComment` (`CommentPlacement` placement) const
Include delimiters and embedded newlines.
- `JSONCPP_STRING` `toStyledString` () const
- `const_iterator` `begin` () const
- `const_iterator` `end` () const
- `iterator` `begin` ()
- `iterator` `end` ()
- void `setOffsetStart` (ptrdiff_t start)
- void `setOffsetLimit` (ptrdiff_t limit)
- ptrdiff_t `getOffsetStart` () const
- ptrdiff_t `getOffsetLimit` () const

Static Public Attributes

- static const `Value` & `null` = reinterpret_cast<const `Value`&>(kNullRef)
We regret this reference to a global instance; prefer the simpler `Value()`.
- static const `Value` & `nullRef` = null
just a kludge for binary-compatibility; same as null
- static const `LargestInt` `minLargestInt` = `LargestInt`(~(`LargestUInt`(-1) / 2))
Minimum signed integer value that can be stored in a `Json::Value`.
- static const `LargestInt` `maxLargestInt` = `LargestInt`(`LargestUInt`(-1) / 2)
Maximum signed integer value that can be stored in a `Json::Value`.
- static const `LargestUInt` `maxLargestUInt` = `LargestUInt`(-1)
Maximum unsigned integer value that can be stored in a `Json::Value`.
- static const `Int` `minInt` = `Int`(~(`UInt`(-1) / 2))
Minimum signed int value that can be stored in a `Json::Value`.
- static const `Int` `maxInt` = `Int`(`UInt`(-1) / 2)
Maximum signed int value that can be stored in a `Json::Value`.
- static const `UInt` `maxUInt` = `UInt`(-1)
Maximum unsigned int value that can be stored in a `Json::Value`.
- static const `Int64` `minInt64`
Minimum signed 64 bits int value that can be stored in a `Json::Value`.
- static const `Int64` `maxInt64`
Maximum signed 64 bits int value that can be stored in a `Json::Value`.
- static const `UInt64` `maxUInt64`
Maximum unsigned 64 bits int value that can be stored in a `Json::Value`.

Private Member Functions

- void `initBasic` (`ValueType` type, bool allocated=false)
- `Value` & `resolveReference` (const char *key)
- `Value` & `resolveReference` (const char *key, const char *end)

Private Attributes

- union `Json::Value::ValueHolder` `value_`
- `ValueType` `type_` : 8
- unsigned int `allocated_` : 1
- `CommentInfo` * `comments_`
- ptrdiff_t `start_`
- ptrdiff_t `limit_`

Friends

- class [ValueIteratorBase](#)

7.45.1 Detailed Description

Represents a [JSON](#) value.

This class is a discriminated union wrapper that can represents a:

- signed integer [range: [Value::minInt](#) - [Value::maxInt](#)]
- unsigned integer (range: 0 - [Value::maxUInt](#))
- double
- UTF-8 string
- boolean
- 'null'
- an ordered list of [Value](#)
- collection of name/value pairs (javascript object)

The type of the held value is represented by a [ValueType](#) and can be obtained using [type\(\)](#).

Values of an [objectValue](#) or [arrayValue](#) can be accessed using [operator\[\]\(\)](#) methods. Non-const methods will automatically create the a [nullValue](#) element if it does not exist. The sequence of an [arrayValue](#) will be automatically resized and initialized with [nullValue](#). [resize\(\)](#) can be used to enlarge or truncate an [arrayValue](#).

The [get\(\)](#) methods can be used to obtain default value in the case the required element does not exist.

It is possible to iterate over the list of a [objectValue](#) values using the [getMemberNames\(\)](#) method.

Note

[Value](#) string-length fit in `size_t`, but keys must be $< 2^{30}$. (The reason is an implementation detail.) A `#↔` `CharReader` will raise an exception if a bound is exceeded to avoid security holes in your app, but the [Value](#) API does *not* check bounds. That is the responsibility of the caller.

7.45.2 Member Typedef Documentation

7.45.2.1 `typedef Json::ArrayIndex Json::Value::ArrayIndex`

7.45.2.2 `typedef ValueConstIterator Json::Value::const_iterator`

7.45.2.3 `typedef Json::Int Json::Value::Int`

7.45.2.4 `typedef Json::Int64 Json::Value::Int64`

7.45.2.5 `typedef ValueIterator Json::Value::iterator`

7.45.2.6 `typedef Json::LargestInt Json::Value::LargestInt`

7.45.2.7 `typedef Json::LargestUInt Json::Value::LargestUInt`

7.45.2.8 `typedef std::vector<JSONCPP_STRING> Json::Value::Members`

7.45.2.9 `typedef std::map<CZString, Value> Json::Value::ObjectValues`

7.45.2.10 `typedef Json::UInt Json::Value::UInt`

7.45.2.11 `typedef Json::UInt64 Json::Value::UInt64`

7.45.3 Constructor & Destructor Documentation

7.45.3.1 `Json::Value::Value (ValueType type = nullValue)`

Create a default [Value](#) of the given type.

This is a very useful constructor. To create an empty array, pass `arrayValue`. To create an empty object, pass `objectValue`. Another [Value](#) can then be set to this one by assignment. This is useful since `clear()` and `resize()` will not alter types.

Examples:

```
Json::Value null_value; // null
Json::Value arr_value(Json::arrayValue); // []
Json::Value obj_value(Json::objectValue); // {}
```

7.45.3.2 `Json::Value::Value (Int value)`

7.45.3.3 `Json::Value::Value (UInt value)`

7.45.3.4 `Json::Value::Value (Int64 value)`

7.45.3.5 `Json::Value::Value (UInt64 value)`

7.45.3.6 `Json::Value::Value (double value)`

7.45.3.7 `Json::Value::Value (const char * value)`

Copy til first 0. (NULL causes to seg-fault.)

7.45.3.8 `Json::Value::Value (const char * begin, const char * end)`

Copy all, incl zeroes.

7.45.3.9 Json::Value::Value (const StaticString & value)

Constructs a value from a static string.

Like other value string constructor but do not duplicate the string for internal storage. The given string must remain alive after the call to this constructor.

Note

This works only for null-terminated strings. (We cannot change the size of this class, so we have nowhere to store the length, which might be computed later for various operations.)

Example of usage:

```
static StaticString foo("some text");  
Json::Value aValue(foo);
```

7.45.3.10 Json::Value::Value (const JSONCPP_STRING & value)

Copy data() til size(). Embedded zeroes too.

7.45.3.11 Json::Value::Value (bool value)

7.45.3.12 Json::Value::Value (const Value & other)

Deep copy.

7.45.3.13 Json::Value::~~Value ()

7.45.4 Member Function Documentation

7.45.4.1 Value & Json::Value::append (const Value & value)

Append value to array at the end.

Equivalent to jsonvalue[jsonvalue.size()] = value;

7.45.4.2 bool Json::Value::asBool () const

7.45.4.3 const char * Json::Value::asCString () const

Embedded zeroes could cause you trouble!

7.45.4.4 `double Json::Value::asDouble () const`

7.45.4.5 `float Json::Value::asFloat () const`

7.45.4.6 `Value::Int Json::Value::asInt () const`

7.45.4.7 `Int64 Json::Value::asInt64 () const`

7.45.4.8 `LargestInt Json::Value::asLargestInt () const`

7.45.4.9 `LargestUInt Json::Value::asLargestUInt () const`

7.45.4.10 `JSONCPP_STRING Json::Value::asString () const`

Embedded zeroes are possible.

7.45.4.11 `Value::UInt Json::Value::asUInt () const`

7.45.4.12 `UInt64 Json::Value::asUInt64 () const`

7.45.4.13 `Value::const_iterator Json::Value::begin () const`

7.45.4.14 `Value::iterator Json::Value::begin ()`

7.45.4.15 `void Json::Value::clear ()`

Remove all object members and array elements.

Precondition

`type()` is `arrayValue`, `objectValue`, or `nullValue`

Postcondition

`type()` is unchanged

7.45.4.16 `int Json::Value::compare (const Value & other) const`

7.45.4.17 `Value const* Json::Value::demand (char const * begin, char const * end)`

Most general and efficient version of object-mutators.

Note

As stated elsewhere, behavior is undefined if $(end - begin) \geq 2^{30}$

Returns

non-zero, but `JSON_ASSERT` if this is neither object nor `nullValue`.

7.45.4.18 **bool** `Json::Value::empty () const`

Return true if empty array, empty object, or null; otherwise, false.

7.45.4.19 **Value::const_iterator** `Json::Value::end () const`

7.45.4.20 **Value::iterator** `Json::Value::end ()`

7.45.4.21 **Value const *** `Json::Value::find (char const * begin, char const * end) const`

Most general and efficient version of `isMember()``const`, `get()``const`, and `operator[]``const`.

Note

As stated elsewhere, behavior is undefined if $(end - begin) \geq 2^{30}$

7.45.4.22 **Value** `Json::Value::get (ArrayIndex index, const Value & defaultValue) const`

If the array contains at least `index+1` elements, returns the element value, otherwise returns `defaultValue`.

7.45.4.23 **Value** `Json::Value::get (const char * key, const Value & defaultValue) const`

Return the member named `key` if it exist, `defaultValue` otherwise.

Note

deep copy

7.45.4.24 **Value** `Json::Value::get (const char * begin, const char * end, const Value & defaultValue) const`

Return the member named `key` if it exist, `defaultValue` otherwise.

Note

deep copy
key may contain embedded nulls.

7.45.4.25 **Value** `Json::Value::get (const JSONCPP_STRING & key, const Value & defaultValue) const`

Return the member named `key` if it exist, `defaultValue` otherwise.

Note

deep copy

Parameters

<i>key</i>	may contain embedded nulls.
------------	-----------------------------

7.45.4.26 **JSONCPP_STRING** `Json::Value::getComment (CommentPlacement placement) const`

Include delimiters and embedded newlines.

7.45.4.27 **Value::Members** `Json::Value::getMemberNames () const`

Return a list of the member names.

If null, return an empty list.

Precondition

`type()` is objectValue or nullValue

Postcondition

if `type()` was nullValue, it remains nullValue

7.45.4.28 **ptrdiff_t** `Json::Value::getOffsetLimit () const`

7.45.4.29 **ptrdiff_t** `Json::Value::getOffsetStart () const`

7.45.4.30 **bool** `Json::Value::getString (char const ** begin, char const ** end) const`

Get raw char* of string-value.

Returns

false if !string. (Seg-fault if str or end are NULL.)

7.45.4.31 `bool Json::Value::hasComment (CommentPlacement placement) const`

7.45.4.32 `void Json::Value::initBasic (ValueType type, bool allocated = false) [private]`

7.45.4.33 `bool Json::Value::isArray () const`

7.45.4.34 `bool Json::Value::isBool () const`

7.45.4.35 `bool Json::Value::isConvertibleTo (ValueType other) const`

7.45.4.36 `bool Json::Value::isDouble () const`

7.45.4.37 `bool Json::Value::isInt () const`

7.45.4.38 `bool Json::Value::isInt64 () const`

7.45.4.39 `bool Json::Value::isIntegral () const`

7.45.4.40 `bool Json::Value::isMember (const char * key) const`

Return true if the object has a member named key.

Note

'key' must be null-terminated.

7.45.4.41 `bool Json::Value::isMember (const JSONCPP_STRING & key) const`

Return true if the object has a member named key.

Parameters

<i>key</i>	may contain embedded nulls.
------------	-----------------------------

7.45.4.42 `bool Json::Value::isMember (const char * begin, const char * end) const`

Same as [isMember\(JSONCPP_STRING const& key\)const](#).

7.45.4.43 `bool Json::Value::isNull () const`

7.45.4.44 `bool Json::Value::isNumeric () const`

7.45.4.45 `bool Json::Value::isObject () const`

7.45.4.46 `bool Json::Value::isString () const`

7.45.4.47 `bool Json::Value::isUInt () const`

7.45.4.48 `bool Json::Value::isUInt64 () const`

7.45.4.49 `bool Json::Value::isValidIndex (ArrayIndex index) const`

Return true if `index < size()`.

7.45.4.50 `bool Json::Value::operator! () const`

Return `isNull()`

7.45.4.51 `bool Json::Value::operator!=(const Value & other) const`

7.45.4.52 `bool Json::Value::operator< (const Value & other) const`

Compare payload only, not comments etc.

7.45.4.53 `bool Json::Value::operator<= (const Value & other) const`

7.45.4.54 `Value & Json::Value::operator= (Value other)`

Deep copy, then `swap(other)`.

Note

Over-write existing comments. To preserve comments, use `swapPayload()`.

7.45.4.55 `bool Json::Value::operator==(const Value & other) const`

7.45.4.56 `bool Json::Value::operator> (const Value & other) const`

7.45.4.57 `bool Json::Value::operator>= (const Value & other) const`

7.45.4.58 `Value & Json::Value::operator[] (ArrayIndex index)`

Access an array element (zero based index).

If the array contains less than `index` element, then null value are inserted in the array so that its size is `index+1`. (You may need to say `'value[0u]'` to get your compiler to distinguish this from the `operator[]` which takes a string.)

7.45.4.59 Value & Json::Value::operator[] (int *index*)

Access an array element (zero based index).

If the array contains less than index element, then null value are inserted in the array so that its size is index+1. (You may need to say 'value[0u]' to get your compiler to distinguish this from the operator[] which takes a string.)

7.45.4.60 const Value & Json::Value::operator[] (ArrayIndex *index*) const

Access an array element (zero based index) (You may need to say 'value[0u]' to get your compiler to distinguish this from the operator[] which takes a string.)

7.45.4.61 const Value & Json::Value::operator[] (int *index*) const

Access an array element (zero based index) (You may need to say 'value[0u]' to get your compiler to distinguish this from the operator[] which takes a string.)

7.45.4.62 Value & Json::Value::operator[] (const char * *key*)

Access an object value by name, create a null member if it does not exist.

Note

Because of our implementation, keys are limited to $2^{30} - 1$ chars. Exceeding that will cause an exception.

7.45.4.63 const Value & Json::Value::operator[] (const char * *key*) const

Access an object value by name, returns null if there is no member with that name.

7.45.4.64 Value & Json::Value::operator[] (const JSONCPP_STRING & *key*)

Access an object value by name, create a null member if it does not exist.

Parameters

<i>key</i>	may contain embedded nulls.
------------	-----------------------------

7.45.4.65 Value const & Json::Value::operator[] (const JSONCPP_STRING & *key*) const

Access an object value by name, returns null if there is no member with that name.

Parameters

<i>key</i>	may contain embedded nulls.
------------	-----------------------------

7.45.4.66 Value & Json::Value::operator[] (const StaticString & key)

Access an object value by name, create a null member if it does not exist.

If the object has no entry for that name, then the member name used to store the new entry is not duplicated.
Example of use:

```
Json::Value object;  
static const StaticString code("code");  
object[code] = 1234;
```

7.45.4.67 bool Json::Value::removeIndex (ArrayIndex i, Value * removed)

Remove the indexed array element.

O(n) expensive operations. Update 'removed' iff removed.

Returns

true iff removed (no exceptions)

7.45.4.68 Value Json::Value::removeMember (const char * key)

Remove and return the named member.

Do nothing if it did not exist.

Returns

the removed Value, or null.

Precondition

type() is objectValue or nullValue

Postcondition

type() is unchanged

Deprecated

7.45.4.69 Value Json::Value::removeMember (const JSONCPP_STRING & key)

Same as removeMember(const char*)

Parameters

<i>key</i>	may contain embedded nulls.
------------	-----------------------------

Deprecated

7.45.4.70 `bool Json::Value::removeMember (const char * key, Value * removed)`

Same as [removeMember\(const char* begin, const char* end, Value* removed\)](#), but 'key' is null-terminated.

7.45.4.71 `bool Json::Value::removeMember (JSONCPP_STRING const & key, Value * removed)`

Remove the named map member.

Update 'removed' iff removed.

Parameters

<i>key</i>	may contain embedded nulls.
------------	-----------------------------

Returns

true iff removed (no exceptions)

7.45.4.72 `bool Json::Value::removeMember (const char * begin, const char * end, Value * removed)`

Same as [removeMember\(JSONCPP_STRING const& key, Value* removed\)](#)

7.45.4.73 `void Json::Value::resize (ArrayIndex size)`

Resize the array to size elements.

New elements are initialized to null. May only be called on nullValue or arrayValue.

Precondition

[type\(\)](#) is arrayValue or nullValue

Postcondition

[type\(\)](#) is arrayValue

7.45.4.74 **Value & Json::Value::resolveReference (const char * *key*)** [private]

7.45.4.75 **Value & Json::Value::resolveReference (const char * *key*, const char * *end*)** [private]

7.45.4.76 **void Json::Value::setComment (const char * *comment*, CommentPlacement *placement*)**

Deprecated Always pass len.

7.45.4.77 **void Json::Value::setComment (const char * *comment*, size_t *len*, CommentPlacement *placement*)**

Comments must be `//... or /* ... */`.

7.45.4.78 **void Json::Value::setComment (const JSONCPP_STRING & *comment*, CommentPlacement *placement*)**

Comments must be `//... or /* ... */`.

7.45.4.79 **void Json::Value::setOffsetLimit (ptrdiff_t *limit*)**

7.45.4.80 **void Json::Value::setOffsetStart (ptrdiff_t *start*)**

7.45.4.81 **ArrayIndex Json::Value::size () const**

Number of values in array or object.

7.45.4.82 **void Json::Value::swap (Value & *other*)**

Swap everything.

7.45.4.83 **void Json::Value::swapPayload (Value & *other*)**

Swap values but leave comments and source offsets in place.

7.45.4.84 **JSONCPP_STRING Json::Value::toStyledString () const**

7.45.4.85 **ValueType Json::Value::type () const**

7.45.5 Friends And Related Function Documentation

7.45.5.1 **friend class ValueIteratorBase** [friend]

7.45.6 Member Data Documentation

7.45.6.1 **unsigned int Json::Value::allocated_** [private]

7.45.6.2 **CommentInfo* Json::Value::comments_** [private]

7.45.6.3 **ptrdiff_t Json::Value::limit_** [private]

7.45.6.4 **const Int Json::Value::maxInt = Int(UInt(-1) / 2)** [static]

Maximum signed int value that can be stored in a [Json::Value](#).

7.45.6.5 `const Int64 Json::Value::maxInt64` `[static]`

Maximum signed 64 bits int value that can be stored in a [Json::Value](#).

7.45.6.6 `const LargestInt Json::Value::maxLargestInt = LargestInt(LargestUInt(-1) / 2)` `[static]`

Maximum signed integer value that can be stored in a [Json::Value](#).

7.45.6.7 `const LargestUInt Json::Value::maxLargestUInt = LargestUInt(-1)` `[static]`

Maximum unsigned integer value that can be stored in a [Json::Value](#).

7.45.6.8 `const UInt Json::Value::maxUInt = UInt(-1)` `[static]`

Maximum unsigned int value that can be stored in a [Json::Value](#).

7.45.6.9 `const UInt64 Json::Value::maxUInt64` `[static]`

Maximum unsigned 64 bits int value that can be stored in a [Json::Value](#).

7.45.6.10 `const Int Json::Value::minInt = Int(~(UInt(-1) / 2))` `[static]`

Minimum signed int value that can be stored in a [Json::Value](#).

7.45.6.11 `const Int64 Json::Value::minInt64` `[static]`

Minimum signed 64 bits int value that can be stored in a [Json::Value](#).

7.45.6.12 `const LargestInt Json::Value::minLargestInt = LargestInt(~(LargestUInt(-1) / 2))` `[static]`

Minimum signed integer value that can be stored in a [Json::Value](#).

7.45.6.13 `const Value & Json::Value::null = reinterpret_cast<const Value&>(kNullRef)` `[static]`

We regret this reference to a global instance; prefer the simpler [Value\(\)](#).

7.45.6.14 `const Value & Json::Value::nullRef = null` `[static]`

just a kludge for binary-compatibility; same as null

7.45.6.15 `ptrdiff_t Json::Value::start_` `[private]`

7.45.6.16 `ValueType Json::Value::type_` `[private]`

7.45.6.17 `union Json::Value::ValueHolder Json::Value::value_` `[private]`

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

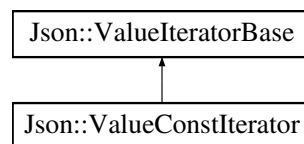
- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.46 Json::ValueConstIterator Class Reference

const iterator for object and array value.

```
#include <json.hpp>
```

Inheritance diagram for `Json::ValueConstIterator`:



Public Types

- typedef const [Value](#) [value_type](#)
- typedef const [Value](#) & [reference](#)
- typedef const [Value](#) * [pointer](#)
- typedef [ValueConstIterator](#) [SelfType](#)
- typedef std::bidirectional_iterator_tag [iterator_category](#)
- typedef unsigned int [size_t](#)
- typedef int [difference_type](#)

Public Member Functions

- [ValueConstIterator](#) ()
- [ValueConstIterator](#) ([ValueIterator](#) const &other)
- [SelfType](#) & operator= (const [ValueIteratorBase](#) &other)
- [SelfType](#) operator++ (int)
- [SelfType](#) operator-- (int)
- [SelfType](#) & operator-- ()
- [SelfType](#) & operator++ ()
- [reference](#) operator* () const
- [pointer](#) operator-> () const
- bool operator== (const [SelfType](#) &other) const
- bool operator!= (const [SelfType](#) &other) const
- [difference_type](#) operator- (const [SelfType](#) &other) const

- [Value key](#) () const
Return either the index or the member name of the referenced value as a [Value](#).
- [UInt index](#) () const
Return the index of the referenced [Value](#), or -1 if it is not an arrayValue.
- [JSONCPP_STRING name](#) () const
Return the member name of the referenced [Value](#), or "" if it is not an objectValue.
- char const * [memberName](#) () const
Return the member name of the referenced [Value](#).
- char const * [memberName](#) (char const **end) const
Return the member name of the referenced [Value](#), or NULL if it is not an objectValue.

Protected Member Functions

- [Value & deref](#) () const
- void [increment](#) ()
- void [decrement](#) ()
- [difference_type computeDistance](#) (const [SelfType](#) &other) const
- bool [isEqual](#) (const [SelfType](#) &other) const
- void [copy](#) (const [SelfType](#) &other)

Private Member Functions

- [ValueConstIterator](#) (const [Value::ObjectValues::iterator](#) ¤t)

Friends

- class [Value](#)

7.46.1 Detailed Description

const iterator for object and array value.

7.46.2 Member Typedef Documentation

7.46.2.1 `typedef int Json::ValueIteratorBase::difference_type [inherited]`

7.46.2.2 `typedef std::bidirectional_iterator_tag Json::ValueIteratorBase::iterator_category [inherited]`

7.46.2.3 `typedef const Value* Json::ValueConstIterator::pointer`

7.46.2.4 `typedef const Value& Json::ValueConstIterator::reference`

7.46.2.5 `typedef ValueConstIterator Json::ValueConstIterator::SelfType`

7.46.2.6 `typedef unsigned int Json::ValueIteratorBase::size_t [inherited]`

7.46.2.7 `typedef const Value Json::ValueConstIterator::value_type`

7.46.3 Constructor & Destructor Documentation

7.46.3.1 `Json::ValueConstIterator::ValueConstIterator ()`

7.46.3.2 `Json::ValueConstIterator::ValueConstIterator (ValueIterator const & other)`

7.46.3.3 `Json::ValueConstIterator::ValueConstIterator (const Value::ObjectValues::iterator & current)` `[explicit]`,
`[private]`

7.46.4 Member Function Documentation

7.46.4.1 `ValueIteratorBase::difference_type Json::ValueIteratorBase::computeDistance (const SelfType & other)`
`const` `[protected]`, `[inherited]`

7.46.4.2 `void Json::ValueIteratorBase::copy (const SelfType & other)` `[protected]`, `[inherited]`

7.46.4.3 `void Json::ValueIteratorBase::decrement ()` `[protected]`, `[inherited]`

7.46.4.4 `Value & Json::ValueIteratorBase::deref () const` `[protected]`, `[inherited]`

7.46.4.5 `void Json::ValueIteratorBase::increment ()` `[protected]`, `[inherited]`

7.46.4.6 `UInt Json::ValueIteratorBase::index () const` `[inherited]`

Return the index of the referenced [Value](#), or -1 if it is not an arrayValue.

7.46.4.7 `bool Json::ValueIteratorBase::isEqual (const SelfType & other) const` `[protected]`, `[inherited]`

7.46.4.8 `Value Json::ValueIteratorBase::key () const` `[inherited]`

Return either the index or the member name of the referenced value as a [Value](#).

7.46.4.9 `char const * Json::ValueIteratorBase::memberName () const` `[inherited]`

Return the member name of the referenced [Value](#).

"" if it is not an objectValue.

Deprecated This cannot be used for UTF-8 strings, since there can be embedded nulls.

7.46.4.10 `char const * Json::ValueIteratorBase::memberName (char const ** end) const` `[inherited]`

Return the member name of the referenced [Value](#), or NULL if it is not an objectValue.

Note

Better version than [memberName\(\)](#). Allows embedded nulls.

7.46.4.11 `JSONCPP_STRING Json::ValueIteratorBase::name () const` `[inherited]`

Return the member name of the referenced [Value](#), or "" if it is not an objectValue.

Note

Avoid `c_str()` on result, as embedded zeroes are possible.

7.46.4.12 `bool Json::ValueIteratorBase::operator!= (const SelfType & other) const` `[inline],[inherited]`

7.46.4.13 `reference Json::ValueConstIterator::operator* () const` `[inline]`

7.46.4.14 `SelfType Json::ValueConstIterator::operator++ (int)` `[inline]`

7.46.4.15 `SelfType& Json::ValueConstIterator::operator++ ()` `[inline]`

7.46.4.16 `difference_type Json::ValueIteratorBase::operator- (const SelfType & other) const` `[inline],[inherited]`

7.46.4.17 `SelfType Json::ValueConstIterator::operator-- (int)` `[inline]`

7.46.4.18 `SelfType& Json::ValueConstIterator::operator-- ()` `[inline]`

7.46.4.19 `pointer Json::ValueConstIterator::operator-> () const` `[inline]`

7.46.4.20 `ValueConstIterator & Json::ValueConstIterator::operator= (const ValueIteratorBase & other)`

7.46.4.21 `bool Json::ValueIteratorBase::operator== (const SelfType & other) const` `[inline],[inherited]`

7.46.5 Friends And Related Function Documentation

7.46.5.1 `friend class Value` `[friend]`

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.47 Json::Value::ValueHolder Union Reference

Public Attributes

- [LargestInt](#) `int_`
- [LargestUInt](#) `uint_`
- `double` `real_`
- `bool` `bool_`
- `char *` `string_`
- [ObjectValues](#) `* map_`

7.47.1 Member Data Documentation

7.47.1.1 `bool` `Json::Value::ValueHolder::bool_`

7.47.1.2 `LargestInt` `Json::Value::ValueHolder::int_`

7.47.1.3 `ObjectValues*` `Json::Value::ValueHolder::map_`

7.47.1.4 `double` `Json::Value::ValueHolder::real_`

7.47.1.5 `char*` `Json::Value::ValueHolder::string_`

7.47.1.6 `LargestUInt` `Json::Value::ValueHolder::uint_`

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

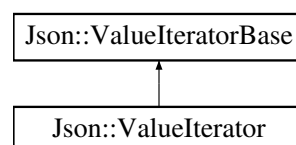
- `/home/pranav/Repositories/zcm/include/json.hpp`

7.48 Json::ValueIterator Class Reference

Iterator for object and array value.

```
#include <json.hpp>
```

Inheritance diagram for `Json::ValueIterator`:



Public Types

- typedef [Value](#) [value_type](#)
- typedef unsigned int [size_t](#)
- typedef int [difference_type](#)
- typedef [Value](#) & [reference](#)
- typedef [Value](#) * [pointer](#)
- typedef [Value](#)iterator [SelfType](#)
- typedef std::bidirectional_iterator_tag [iterator_category](#)

Public Member Functions

- [Value](#)iterator ()
- [Value](#)iterator (const [Value](#)ConstIterator &other)
- [Value](#)iterator (const [Value](#)iterator &other)
- [SelfType](#) & operator= (const [SelfType](#) &other)
- [SelfType](#) operator++ (int)
- [SelfType](#) operator-- (int)
- [SelfType](#) & operator-- ()
- [SelfType](#) & operator++ ()
- [reference](#) operator* () const
- [pointer](#) operator-> () const
- bool operator== (const [SelfType](#) &other) const
- bool operator!= (const [SelfType](#) &other) const
- [difference_type](#) operator- (const [SelfType](#) &other) const
- [Value](#) key () const
Return either the index or the member name of the referenced value as a [Value](#).
- [UInt](#) index () const
Return the index of the referenced [Value](#), or -1 if it is not an arrayValue.
- [JSONCPP_STRING](#) name () const
Return the member name of the referenced [Value](#), or "" if it is not an objectValue.
- char const * [memberName](#) () const
Return the member name of the referenced [Value](#).
- char const * [memberName](#) (char const **end) const
Return the member name of the referenced [Value](#), or NULL if it is not an objectValue.

Protected Member Functions

- [Value](#) & deref () const
- void increment ()
- void decrement ()
- [difference_type](#) computeDistance (const [SelfType](#) &other) const
- bool isEqual (const [SelfType](#) &other) const
- void copy (const [SelfType](#) &other)

Private Member Functions

- [Value](#)iterator (const [Value](#)::ObjectValues::iterator ¤t)

Friends

- class [Value](#)

7.48.1 Detailed Description

Iterator for object and array value.

7.48.2 Member Typedef Documentation

7.48.2.1 `typedef int Json::ValueIterator::difference_type`

7.48.2.2 `typedef std::bidirectional_iterator_tag Json::ValueIteratorBase::iterator_category` [inherited]

7.48.2.3 `typedef Value* Json::ValueIterator::pointer`

7.48.2.4 `typedef Value& Json::ValueIterator::reference`

7.48.2.5 `typedef ValueIterator Json::ValueIterator::SelfType`

7.48.2.6 `typedef unsigned int Json::ValueIterator::size_t`

7.48.2.7 `typedef Value Json::ValueIterator::value_type`

7.48.3 Constructor & Destructor Documentation

7.48.3.1 `Json::ValueIterator::ValueIterator ()`

7.48.3.2 `Json::ValueIterator::ValueIterator (const ValueConstIterator & other)` [explicit]

7.48.3.3 `Json::ValueIterator::ValueIterator (const ValueIterator & other)`

7.48.3.4 `Json::ValueIterator::ValueIterator (const Value::ObjectValues::iterator & current)` [explicit],[private]

7.48.4 Member Function Documentation

7.48.4.1 `ValueIteratorBase::difference_type Json::ValueIteratorBase::computeDistance (const SelfType & other) const` [protected],[inherited]

7.48.4.2 `void Json::ValueIteratorBase::copy (const SelfType & other)` [protected],[inherited]

7.48.4.3 `void Json::ValueIteratorBase::decrement ()` [protected],[inherited]

7.48.4.4 `Value & Json::ValueIteratorBase::deref () const` [protected],[inherited]

7.48.4.5 `void Json::ValueIteratorBase::increment ()` [protected],[inherited]

7.48.4.6 `UInt Json::ValueIteratorBase::index () const` [inherited]

Return the index of the referenced [Value](#), or -1 if it is not an arrayValue.

7.48.4.7 `bool Json::ValueIteratorBase::isEqual (const SelfType & other) const` [protected],[inherited]

7.48.4.8 `Value Json::ValueIteratorBase::key () const` [inherited]

Return either the index or the member name of the referenced value as a [Value](#).

7.48.4.9 `char const * Json::ValueIteratorBase::memberName () const` [inherited]

Return the member name of the referenced [Value](#).

"" if it is not an objectValue.

Deprecated This cannot be used for UTF-8 strings, since there can be embedded nulls.

7.48.4.10 `char const * Json::ValueIteratorBase::memberName (char const ** end) const` [inherited]

Return the member name of the referenced [Value](#), or NULL if it is not an objectValue.

Note

Better version than [memberName\(\)](#). Allows embedded nulls.

7.48.4.11 `JSONCPP_STRING Json::ValueIteratorBase::name () const` [inherited]

Return the member name of the referenced [Value](#), or "" if it is not an objectValue.

Note

Avoid `c_str()` on result, as embedded zeroes are possible.

7.48.4.12 `bool Json::ValueIteratorBase::operator!= (const SelfType & other) const` [inline],[inherited]

7.48.4.13 `reference Json::ValueIterator::operator* () const` [inline]

7.48.4.14 `SelfType Json::ValueIterator::operator++ (int)` [inline]

7.48.4.15 `SelfType& Json::ValueIterator::operator++ ()` [inline]

7.48.4.16 `difference_type Json::ValueIteratorBase::operator- (const SelfType & other) const` [inline],[inherited]

7.48.4.17 `SelfType Json::ValueIterator::operator-- (int)` [inline]

7.48.4.18 `SelfType& Json::ValueIterator::operator-- ()` [inline]

7.48.4.19 `pointer Json::ValueIterator::operator-> () const` [inline]

7.48.4.20 `ValueIterator & Json::ValueIterator::operator= (const SelfType & other)`

7.48.4.21 `bool Json::ValueIteratorBase::operator== (const SelfType & other) const` [inline],[inherited]

7.48.5 Friends And Related Function Documentation

7.48.5.1 `friend class Value` [friend]

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

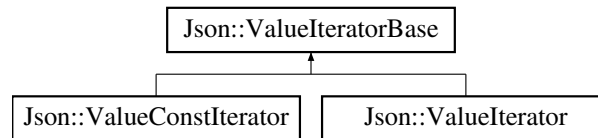
- `/home/pranav/Repositories/zcm/include/json.hpp`
- `/home/pranav/Repositories/zcm/src/json.cpp`

7.49 Json::ValueIteratorBase Class Reference

base class for [Value](#) iterators.

```
#include <json.hpp>
```

Inheritance diagram for Json::ValueIteratorBase:



Public Types

- typedef std::bidirectional_iterator_tag [iterator_category](#)
- typedef unsigned int [size_t](#)
- typedef int [difference_type](#)
- typedef [ValueIteratorBase](#) SelfType

Public Member Functions

- bool [operator==](#) (const SelfType &other) const
- bool [operator!=](#) (const SelfType &other) const
- [difference_type](#) [operator-](#) (const SelfType &other) const
- [Value](#) [key](#) () const
Return either the index or the member name of the referenced value as a [Value](#).
- [UInt](#) [index](#) () const
Return the index of the referenced [Value](#), or -1 if it is not an arrayValue.
- [JSONCPP_STRING](#) [name](#) () const
Return the member name of the referenced [Value](#), or "" if it is not an objectValue.
- char const * [memberName](#) () const
Return the member name of the referenced [Value](#).
- char const * [memberName](#) (char const **end) const
Return the member name of the referenced [Value](#), or NULL if it is not an objectValue.
- [ValueIteratorBase](#) ()
- [ValueIteratorBase](#) (const [Value::ObjectValues::iterator](#) ¤t)

Protected Member Functions

- [Value](#) & [deref](#) () const
- void [increment](#) ()
- void [decrement](#) ()
- [difference_type](#) [computeDistance](#) (const SelfType &other) const
- bool [isEqual](#) (const SelfType &other) const
- void [copy](#) (const SelfType &other)

Private Attributes

- Value::ObjectValues::iterator [current_](#)
- bool [isNull_](#)

7.49.1 Detailed Description

base class for [Value](#) iterators.

7.49.2 Member Typedef Documentation

7.49.2.1 `typedef int Json::ValueIteratorBase::difference_type`

7.49.2.2 `typedef std::bidirectional_iterator_tag Json::ValueIteratorBase::iterator_category`

7.49.2.3 `typedef ValueIteratorBase Json::ValueIteratorBase::SelfType`

7.49.2.4 `typedef unsigned int Json::ValueIteratorBase::size_t`

7.49.3 Constructor & Destructor Documentation

7.49.3.1 `Json::ValueIteratorBase::ValueIteratorBase ()`

7.49.3.2 `Json::ValueIteratorBase::ValueIteratorBase (const Value::ObjectValues::iterator & current)` `[explicit]`

7.49.4 Member Function Documentation

7.49.4.1 `ValueIteratorBase::difference_type Json::ValueIteratorBase::computeDistance (const SelfType & other)`
`const` `[protected]`

7.49.4.2 `void Json::ValueIteratorBase::copy (const SelfType & other)` `[protected]`

7.49.4.3 `void Json::ValueIteratorBase::decrement ()` `[protected]`

7.49.4.4 `Value & Json::ValueIteratorBase::deref () const` `[protected]`

7.49.4.5 `void Json::ValueIteratorBase::increment ()` `[protected]`

7.49.4.6 `UInt Json::ValueIteratorBase::index () const`

Return the index of the referenced [Value](#), or -1 if it is not an arrayValue.

7.49.4.7 `bool Json::ValueIteratorBase::isEqual (const SelfType & other) const` `[protected]`

7.49.4.8 `Value Json::ValueIteratorBase::key () const`

Return either the index or the member name of the referenced value as a [Value](#).

7.49.4.9 `char const * Json::ValueIteratorBase::memberName () const`

Return the member name of the referenced [Value](#).

"" if it is not an objectValue.

Deprecated This cannot be used for UTF-8 strings, since there can be embedded nulls.

7.49.4.10 `char const * Json::ValueIteratorBase::memberName (char const ** end) const`

Return the member name of the referenced [Value](#), or NULL if it is not an objectValue.

Note

Better version than [memberName\(\)](#). Allows embedded nulls.

7.49.4.11 `JSONCPP_STRING Json::ValueIteratorBase::name () const`

Return the member name of the referenced [Value](#), or "" if it is not an objectValue.

Note

Avoid `c_str()` on result, as embedded zeroes are possible.

7.49.4.12 `bool Json::ValueIteratorBase::operator!= (const SelfType & other) const` `[inline]`

7.49.4.13 `difference_type Json::ValueIteratorBase::operator- (const SelfType & other) const` `[inline]`

7.49.4.14 `bool Json::ValueIteratorBase::operator== (const SelfType & other) const` `[inline]`

7.49.5 Member Data Documentation

7.49.5.1 `Value::ObjectValues::iterator Json::ValueIteratorBase::current_` `[private]`

7.49.5.2 `bool Json::ValueIteratorBase::isNull_` `[private]`

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

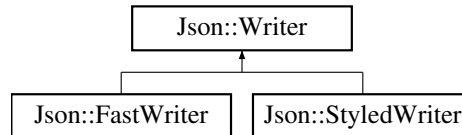
- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

7.50 Json::Writer Class Reference

Abstract class for writers.

```
#include <json.hpp>
```

Inheritance diagram for Json::Writer:



Public Member Functions

- virtual [~Writer](#) ()
- virtual [JSONCPP_STRING write](#) (const [Value](#) &root)=0

7.50.1 Detailed Description

Abstract class for writers.

Deprecated Use [StreamWriter](#). (And really, this is an implementation detail.)

7.50.2 Constructor & Destructor Documentation

7.50.2.1 [Json::Writer::~~Writer](#) () `[virtual]`

7.50.3 Member Function Documentation

7.50.3.1 `virtual JSONCPP_STRING Json::Writer::write (const Value & root)` `[pure virtual]`

Implemented in [Json::StyledWriter](#), and [Json::FastWriter](#).

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

- [/home/pranav/Repositories/zcm/include/json.hpp](#)
- [/home/pranav/Repositories/zcm/src/json.cpp](#)

Chapter 8

File Documentation

8.1 /home/pranav/Repositories/zcm/include/actor.hpp File Reference

This file declares the Actor class.

```
#include "json.hpp"
#include "component.hpp"
#include <dlfcn.h>
#include <fstream>
```

Classes

- class [zcm::Actor](#)
Actor class.

Namespaces

- [zcm](#)

8.1.1 Detailed Description

This file declares the Actor class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.2 /home/pranav/Repositories/zcm/include/client.hpp File Reference

This file declares the Client class.

```
#include <iostream>
#include <zmq.hpp>
```

Classes

- class [zcm::Client](#)
Client class.

Namespaces

- [zcm](#)

8.2.1 Detailed Description

This file declares the Client class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.3 /home/pranav/Repositories/zcm/include/component.hpp File Reference

This file declares the Component class.

```
#include "timer.hpp"
#include "publisher.hpp"
#include "subscriber.hpp"
#include "client.hpp"
#include "server.hpp"
```

Classes

- class [zcm::Component](#)
Component class.

Namespaces

- [zcm](#)

8.3.1 Detailed Description

This file declares the Component class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.4 /home/pranav/Repositories/zcm/include/json.hpp File Reference

```
#include <stddef.h>
#include <string>
#include <vector>
#include <exception>
#include <map>
#include <deque>
#include <iosfwd>
#include <stack>
#include <istream>
#include <ostream>
#include <stdlib.h>
#include <sstream>
```

Classes

- class [Json::Features](#)
Configuration passed to reader and writer.
- class [Json::Exception](#)
Base class for all exceptions we throw.
- class [Json::RuntimeError](#)
Exceptions which the user cannot easily avoid.
- class [Json::LogicError](#)
Exceptions thrown by JSON_ASSERT/JSON_FAIL macros.
- class [Json::StaticString](#)
Lightweight wrapper to tag static string.
- class [Json::Value](#)
Represents a JSON value.
- class [Json::Value::CZString](#)
- struct [Json::Value::CZString::StringStorage](#)
- struct [Json::Value::CommentInfo](#)
- union [Json::Value::ValueHolder](#)

- class [Json::PathArgument](#)
Experimental and untested: represents an element of the "path" to access a node.
- class [Json::Path](#)
Experimental and untested: represents a "path" to access a node.
- class [Json::ValueIteratorBase](#)
base class for [Value](#) iterators.
- class [Json::ValueConstIterator](#)
const iterator for object and array value.
- class [Json::ValueIterator](#)
Iterator for object and array value.
- class [Json::Reader](#)
Unserialize a [JSON](#) document into a [Value](#).
- struct [Json::Reader::StructuredError](#)
An error tagged with where in the JSON text it was encountered.
- class [Json::Reader::Token](#)
- class [Json::Reader::ErrorInfo](#)
- class [Json::CharReader](#)
Interface for reading JSON from a char array.
- class [Json::CharReader::Factory](#)
- class [Json::CharReaderBuilder](#)
Build a [CharReader](#) implementation.
- class [Json::StreamWriter](#)
Usage:
- class [Json::StreamWriter::Factory](#)
A simple abstract factory.
- class [Json::StreamWriterBuilder](#)
Build a [StreamWriter](#) implementation.
- class [Json::Writer](#)
Abstract class for writers.
- class [Json::FastWriter](#)
Outputs a [Value](#) in [JSON](#) format without formatting (not human friendly).
- class [Json::StyledWriter](#)
Writes a [Value](#) in [JSON](#) format in a human friendly way.
- class [Json::StyledStreamWriter](#)
Writes a [Value](#) in [JSON](#) format in a human friendly way, to a stream rather than to a string.

Namespaces

- [Json](#)
JSON (JavaScript Object Notation).
- [std](#)

Macros

- #define `JSON_IS_AMALGAMATION`
Json-cpp amalgated header (<http://jsoncpp.sourceforge.net/>).
- #define `JSON_VERSION_H_INCLUDED`
- #define `JSONCPP_VERSION_STRING` "1.7.2"
- #define `JSONCPP_VERSION_MAJOR` 1
- #define `JSONCPP_VERSION_MINOR` 7
- #define `JSONCPP_VERSION_PATCH` 2
- #define `JSONCPP_VERSION_QUALIFIER`
- #define `JSONCPP_VERSION_HEX` ((`JSONCPP_VERSION_MAJOR` << 24) | (`JSONCPP_VERSION_MINOR` << 16) | (`JSONCPP_VERSION_PATCH` << 8))
- #define `JSONCPP_USING_SECURE_MEMORY` 0
- #define `JSON_CONFIG_H_INCLUDED`
- #define `JSON_USE_EXCEPTION` 1
If defined, indicates that json library is embedded in CppTL library.
- #define `JSON_API`
If defined, indicates that the source file is amalgated to prevent private header inclusion.
- #define `JSONCPP_OVERRIDE` override
- #define `JSON_HAS_RVALUE_REFERENCES` 0
- #define `JSONCPP_DEPRECATED`(message)
- #define `JSON_HAS_INT64`
- #define `JSONCPP_STRING` std::string
- #define `JSONCPP_OSTRINGSTREAM` std::ostringstream
- #define `JSONCPP_OSTREAM` std::ostream
- #define `JSONCPP_ISTRINGSTREAM` std::istream
- #define `JSONCPP_ISTREAM` std::istream
- #define `JSON_FORWARDS_H_INCLUDED`
- #define `CPPTL_JSON_FEATURES_H_INCLUDED`
- #define `CPPTL_JSON_H_INCLUDED`
- #define `JSONCPP_NORETURN`
- #define `CPPTL_JSON_READER_H_INCLUDED`
- #define `JSON_WRITER_H_INCLUDED`
- #define `CPPTL_JSON_ASSERTIONS_H_INCLUDED`
- #define `JSON_ASSERT`(condition) {if (!(condition)) {`Json::throwLogicError`("assert json failed");}}
- It should not be possible for a maliciously designed file to cause an abort() or seg-fault, so these macros are used only for pre-condition violations and internal logic errors.*
- #define `JSON_FAIL_MESSAGE`(message)
- #define `JSON_ASSERT_MESSAGE`(condition, message)

Typedefs

- typedef int `Json::Int`
- typedef unsigned int `Json::UInt`
- typedef long long int `Json::Int64`
- typedef unsigned long long int `Json::UInt64`
- typedef Int64 `Json::LargestInt`
- typedef UInt64 `Json::LargestUInt`
- typedef unsigned int `Json::ArrayIndex`

Enumerations

- enum `Json::ValueType` {
`Json::nullValue = 0, Json::intValue, Json::uintValue, Json::realValue,`
`Json::stringValue, Json::booleanValue, Json::arrayValue, Json::objectValue` }
Type of the value held by a Value object.
- enum `Json::CommentPlacement` { `Json::commentBefore = 0, Json::commentAfterOnSameLine, Json::commentAfter, Json::numberOfCommentPlacement` }

Functions

- `JSONCPP_NORETURN void Json::throwRuntimeError (JSONCPP_STRING const &msg)`
used internally
- `JSONCPP_NORETURN void Json::throwLogicError (JSONCPP_STRING const &msg)`
used internally
- `template<>`
`void std::swap (Json::Value &a, Json::Value &b)`
Specialize std::swap() for Json::Value.
- `bool JSON_API Json::parseFromStream (CharReader::Factory const &, JSONCPP_ISTREAM &, Value *root, std::string *errs)`
Consume entire stream and use its begin/end.
- `JSONCPP_ISTREAM & Json::operator>> (JSONCPP_ISTREAM &, Value &)`
Read from 'sin' into 'root'.
- `JSONCPP_STRING Json::writeString (StreamWriter::Factory const &factory, Value const &root)`
Write into stringstream, then return string, for convenience.
- `JSONCPP_STRING JSON_API Json::valueToString (Int value)`
- `JSONCPP_STRING JSON_API Json::valueToString (UInt value)`
- `JSONCPP_STRING Json::valueToString (LargestInt value)`
- `JSONCPP_STRING Json::valueToString (LargestUInt value)`
- `JSONCPP_STRING Json::valueToString (double value)`
- `JSONCPP_STRING Json::valueToString (bool value)`
- `JSONCPP_STRING Json::valueToQuotedString (const char *value)`
- `JSONCPP_OSTREAM & Json::operator<< (JSONCPP_OSTREAM &, const Value &root)`
Output using the StyledStreamWriter.

8.4.1 Macro Definition Documentation

8.4.1.1 `#define CPPTL_JSON_ASSERTIONS_H_INCLUDED`

8.4.1.2 `#define CPPTL_JSON_FEATURES_H_INCLUDED`

8.4.1.3 `#define CPPTL_JSON_H_INCLUDED`

8.4.1.4 `#define CPPTL_JSON_READER_H_INCLUDED`

8.4.1.5 `#define JSON_API`

If defined, indicates that the source file is amalgated to prevent private header inclusion.

Remarks: it is automatically defined in the generated amalgated header.

8.4.1.6 `#define JSON_ASSERT(condition) {if (!(condition)) {Json::throwLogicError("assert json failed");}}`

It should not be possible for a maliciously designed file to cause an abort() or seg-fault, so these macros are used only for pre-condition violations and internal logic errors.

8.4.1.7 `#define JSON_ASSERT_MESSAGE(condition, message)`

Value:

```
if (!(condition)) {
    JSON_FAIL_MESSAGE(message);
}
```

8.4.1.8 `#define JSON_CONFIG_H_INCLUDED`

8.4.1.9 `#define JSON_FAIL_MESSAGE(message)`

Value:

```
{
    JSONCPP_OSTRINGSTREAM oss; oss << message;
    Json::throwLogicError(oss.str());
    abort();
}
```

8.4.1.10 `#define JSON_FORWARDS_H_INCLUDED`

8.4.1.11 `#define JSON_HAS_INT64`

8.4.1.12 `#define JSON_HAS_RVALUE_REFERENCES 0`

8.4.1.13 `#define JSON_IS_AMALGAMATION`

Json-cpp amalgated header (<http://jsoncpp.sourceforge.net/>).

It is intended to be used with `#include "json/json.h"` If defined, indicates that the source file is amalgated to prevent private header inclusion.

8.4.1.14 `#define JSON_USE_EXCEPTION 1`

If defined, indicates that json library is embedded in CppTL library.

If defined, indicates that json may leverage CppTL library If defined, indicates that cpptl vector based map should be used instead of `std::map` as Value container.

8.4.1.15 `#define JSON_VERSION_H_INCLUDED`

8.4.1.16 `#define JSON_WRITER_H_INCLUDED`

8.4.1.17 `#define JSONCPP_DEPRECATED(message)`

8.4.1.18 `#define JSONCPP_ISTREAM std::istream`

8.4.1.19 `#define JSONCPP_ISTRINGSTREAM std::istringstream`

8.4.1.20 `#define JSONCPP_NORETURN`

8.4.1.21 `#define JSONCPP_OSTREAM std::ostream`

8.4.1.22 `#define JSONCPP_OSTRINGSTREAM std::ostringstream`

8.4.1.23 `#define JSONCPP_OVERRIDE override`

8.4.1.24 `#define JSONCPP_STRING std::string`

8.4.1.25 `#define JSONCPP_USING_SECURE_MEMORY 0`

8.4.1.26 `#define JSONCPP_VERSION_HEX ((JSONCPP_VERSION_MAJOR << 24) | (JSONCPP_VERSION_MINOR << 16) | (JSONCPP_VERSION_PATCH << 8))`

8.4.1.27 `#define JSONCPP_VERSION_MAJOR 1`

8.4.1.28 `#define JSONCPP_VERSION_MINOR 7`

8.4.1.29 `#define JSONCPP_VERSION_PATCH 2`

8.4.1.30 `#define JSONCPP_VERSION_QUALIFIER`

8.4.1.31 `#define JSONCPP_VERSION_STRING "1.7.2"`

8.5 `/home/pranav/Repositories/zcm/include/operation_queue.hpp` File Reference

This file declares the `Operation_Queue` class.

```
#include <iostream>
#include <queue>
#include <mutex>
#include <thread>
#include <functional>
#include "operation_types.hpp"
```


Classes

- class [zcm::Operation_Queue](#)
Operation_Queue class.
- struct [zcm::Operation_Queue::PriorityOrdering](#)

Namespaces

- [zcm](#)

8.5.1 Detailed Description

This file declares the `Operation_Queue` class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.6 /home/pranav/Repositories/zcm/include/operation_types.hpp File Reference

This file declares Operation Types.

```
#include <iostream>
#include <functional>
#include "zmq.hpp"
```

Classes

- class [zcm::Base_Operation](#)
Base Operation class.
- class [zcm::Timer_Operation](#)
Timer Operation class.
- class [zcm::Subscriber_Operation](#)
Subscriber Operation class.
- class [zcm::Server_Operation](#)
Server Operation class.

Namespaces

- [zcm](#)

8.6.1 Detailed Description

This file declares Operation Types.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.7 /home/pranav/Repositories/zcm/include/publisher.hpp File Reference

This file declares the Publisher class.

```
#include <iostream>
#include <zmq.hpp>
```

Classes

- class [zcm::Publisher](#)
Publisher class.

Namespaces

- [zcm](#)

8.7.1 Detailed Description

This file declares the Publisher class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.8 /home/pranav/Repositories/zcm/include/server.hpp File Reference

This file declares the Server class.

```
#include <iostream>
#include <vector>
#include <map>
#include <sstream>
#include <zmq.hpp>
#include "operation_queue.hpp"
```

Classes

- class [zcm::Server](#)
Server class.

Namespaces

- [zcm](#)

8.8.1 Detailed Description

This file declares the Server class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.9 /home/pranav/Repositories/zcm/include/subscriber.hpp File Reference

This file declares the Subscriber class.

```
#include <iostream>
#include <vector>
#include <map>
#include <sstream>
#include <zmq.hpp>
#include "operation_queue.hpp"
```

Classes

- class [zcm::Subscriber](#)
Subscriber class.

Namespaces

- [zcm](#)

8.9.1 Detailed Description

This file declares the Subscriber class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.10 /home/pranav/Repositories/zcm/include/timer.hpp File Reference

This file declares the Timer class.

```
#include <iostream>
#include <string>
#include <chrono>
#include <ratio>
#include <thread>
#include "operation_queue.hpp"
```

Classes

- class [zcm::Timer](#)
Timer class.

Namespaces

- [zcm](#)

8.10.1 Detailed Description

This file declares the Timer class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.11 /home/pranav/Repositories/zcm/src/actor.cpp File Reference

This file contains definitions for the Actor class.

```
#include "actor.hpp"
```

Namespaces

- [zcm](#)

8.11.1 Detailed Description

This file contains definitions for the Actor class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.12 /home/pranav/Repositories/zcm/src/client.cpp File Reference

This file contains definitions for the Client class.

```
#include "client.hpp"
```

Namespaces

- [zcm](#)

8.12.1 Detailed Description

This file contains definitions for the Client class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.13 /home/pranav/Repositories/zcm/src/component.cpp File Reference

This file contains definitions for the Component class.

```
#include "component.hpp"
```

Namespaces

- [zcm](#)

8.13.1 Detailed Description

This file contains definitions for the Component class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.14 /home/pranav/Repositories/zcm/src/json.cpp File Reference

```
#include "json.hpp"
#include <json/assertions.h>
#include <json/reader.h>
#include <json/value.h>
#include "json_tool.h"
#include <utility>
#include <cstdio>
#include <cassert>
#include <cstring>
#include <istream>
#include <sstream>
#include <memory>
#include <set>
#include <limits>
#include <json/writer.h>
#include <math.h>
#include <cstdint>
#include <algorithm>
#include "json_valueiterator.inl"
#include <iomanip>
#include <cmath>
```

Classes

- class [Json::OurFeatures](#)
 - class [Json::OurReader](#)
 - struct [Json::OurReader::StructuredError](#)
 - class [Json::OurReader::Token](#)
 - class [Json::OurReader::ErrorInfo](#)
 - class [Json::OurCharReader](#)
 - struct [Json::CommentStyle](#)
 - struct [Json::BuiltStyledStreamWriter](#)
- Scoped enums are not available until C++11.*

Namespaces

- [Json](#)
- JSON (JavaScript Object Notation).*

Macros

- `#define` [LIB_JSONCPP_JSON_TOOL_H_INCLUDED](#)
- *Json-cpp amalgated source (<http://jsoncpp.sourceforge.net/>).*
- `#define` [JSON_ASSERT_UNREACHABLE](#) `assert(false)`
- `#define` [ALIGNAS](#)(`byte_alignment`)
- `#define` [isfinite](#) `std::isfinite`

Typedefs

- `typedef` `char` [Json::UIntToStringBuffer](#)[`UIntToStringBufferSize`]
- `typedef` `std::auto_ptr< CharReader >` [Json::CharReaderPtr](#)
- `typedef` `std::auto_ptr< StreamWriter >` [Json::StreamWriterPtr](#)

Enumerations

- `enum` { [Json::uintToStringBufferSize](#) = 3 * `sizeof(LargestUInt)` + 1 }

Functions

- `static` [JSONCPP_STRING](#) [Json::codePointToUTF8](#) (`unsigned int` cp)
- *Converts a unicode code-point to UTF-8.*
- `static` `bool` [Json::isControlCharacter](#) (`char` ch)
- *Returns true if ch is a control character (in range [1,31]).*
- `static` `void` [Json::uintToString](#) (`LargestUInt` value, `char` *¤t)
- *Converts an unsigned integer to string.*
- `static` `void` [Json::fixNumericLocale](#) (`char` *begin, `char` *end)
- *Change ',' to '.*
- `static` `bool` [Json::containsNewLine](#) (`Reader::Location` begin, `Reader::Location` end)
- `static` [JSONCPP_STRING](#) [Json::normalizeEOL](#) (`Reader::Location` begin, `Reader::Location` end)
- `static` `void` [Json::getValidReaderKeys](#) (`std::set< JSONCPP_STRING >` *valid_keys)

- `bool Json::parseFromStream` (CharReader::Factory const &fact, JSONCPP_ISTREAM &sin, Value *root, JSONCPP_STRING *errs)
- `JSONCPP_ISTREAM & Json::operator>>` (JSONCPP_ISTREAM &, Value &)
 - Read from 'sin' into 'root'.*
- `static const unsigned char Json::ALIGNAS` (8) kNull[sizeof(Value)]
- `template<typename T, typename U >`
 - `static bool Json::InRange` (double d, T min, U max)
- `static char * Json::duplicateStringValue` (const char *value, size_t length)
 - Duplicates the specified string value.*
- `static char * Json::duplicateAndPrefixStringValue` (const char *value, unsigned int length)
- `static void Json::decodePrefixedString` (bool isPrefixed, char const *prefixed, unsigned *length, char const **value)
- `static void Json::releasePrefixedStringValue` (char *value)
 - Free the string duplicated by `duplicateStringValue()`/`duplicateAndPrefixStringValue()`.*
- `static void Json::releaseStringValue` (char *value, unsigned)
- `JSONCPP_NORETURN void Json::throwRuntimeError` (JSONCPP_STRING const &msg)
 - used internally*
- `JSONCPP_NORETURN void Json::throwLogicError` (JSONCPP_STRING const &msg)
 - used internally*
- `static bool Json::IsIntegral` (double d)
- `static bool Json::containsControlCharacter` (const char *str)
- `static bool Json::containsControlCharacter0` (const char *str, unsigned len)
- `JSONCPP_STRING Json::valueToString` (LargestInt value)
- `JSONCPP_STRING Json::valueToString` (LargestUInt value)
- `JSONCPP_STRING Json::valueToString` (double value, bool useSpecialFloats, unsigned int precision)
- `JSONCPP_STRING Json::valueToString` (double value)
- `JSONCPP_STRING Json::valueToString` (bool value)
- `JSONCPP_STRING Json::valueToQuotedString` (const char *value)
- `static char const * Json::strnpbrk` (char const *s, char const *accept, size_t n)
- `static JSONCPP_STRING Json::valueToQuotedStringN` (const char *value, unsigned length)
- `static void Json::getValidWriterKeys` (std::set< JSONCPP_STRING > *valid_keys)
- `JSONCPP_STRING Json::writeString` (StreamWriter::Factory const &factory, Value const &root)
 - Write into stringstream, then return string, for convenience.*
- `JSONCPP_OSTREAM & Json::operator<<` (JSONCPP_OSTREAM &, const Value &root)
 - Output using the `StyledStreamWriter`.*

Variables

- `static int const stackLimit_g` = 1000
- `static int stackDepth_g` = 0
- `const unsigned char & Json::kNullRef` = kNull[0]

8.14.1 Macro Definition Documentation

8.14.1.1 `#define ALIGNAS(byte_alignment)`

8.14.1.2 `#define isfinite std::isfinite`

8.14.1.3 `#define JSON_ASSERT_UNREACHABLE assert(false)`

8.14.1.4 `#define LIB_JSONCPP_JSON_TOOL_H_INCLUDED`

Json-cpp amalgated source (<http://jsoncpp.sourceforge.net/>).

It is intended to be used with `#include "json/json.h"`

8.14.2 Variable Documentation

8.14.2.1 `int stackDepth_g = 0` `[static]`

8.14.2.2 `int const stackLimit_g = 1000` `[static]`

8.15 /home/pranav/Repositories/zcm/src/operation_queue.cpp File Reference

This file contains definitions for the `Operation_Queue` class.

```
#include "operation_queue.hpp"
```

Namespaces

- [zcm](#)

8.15.1 Detailed Description

This file contains definitions for the `Operation_Queue` class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.16 /home/pranav/Repositories/zcm/src/operation_types.cpp File Reference

This file contains definitions for various Operation Types.

```
#include "operation_types.hpp"
```

Namespaces

- [zcm](#)

8.16.1 Detailed Description

This file contains definitions for various Operation Types.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.17 /home/pranav/Repositories/zcm/src/publisher.cpp File Reference

This file contains definitions for the Publisher class.

```
#include "publisher.hpp"
```

Namespaces

- [zcm](#)

8.17.1 Detailed Description

This file contains definitions for the Publisher class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.18 /home/pranav/Repositories/zcm/src/server.cpp File Reference

This file contains definitions for the Server class.

```
#include "server.hpp"
```

Namespaces

- [zcm](#)

8.18.1 Detailed Description

This file contains definitions for the Server class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.19 /home/pranav/Repositories/zcm/src/subscriber.cpp File Reference

This file contains definitions for the Subscriber class.

```
#include "subscriber.hpp"
```

Namespaces

- [zcm](#)

8.19.1 Detailed Description

This file contains definitions for the Subscriber class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

8.20 /home/pranav/Repositories/zcm/src/timer.cpp File Reference

This file contains definitions for the Timer class.

```
#include "timer.hpp"
```

Namespaces

- [zcm](#)

8.20.1 Detailed Description

This file contains definitions for the Timer class.

Author

Pranav Srinivas Kumar

Date

2016.04.24

