

xapian-core  
1.2.6

Generated by Doxygen 1.5.9

Mon Jun 13 01:01:43 2011



# Contents

<b>1</b>	<b>Deprecated List</b>	<b>1</b>
<b>2</b>	<b>Namespace Index</b>	<b>3</b>
2.1	Namespace List . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11
<b>6</b>	<b>Namespace Documentation</b>	<b>13</b>
6.1	Xpian Namespace Reference . . . . .	13
6.1.1	Detailed Description . . . . .	21
6.1.2	Typedef Documentation . . . . .	21
6.1.2.1	doccount . . . . .	21
6.1.2.2	doccount_diff . . . . .	21
6.1.2.3	docid . . . . .	21
6.1.2.4	doclength . . . . .	21
6.1.2.5	percent . . . . .	22
6.1.2.6	termcount . . . . .	22
6.1.2.7	termcount_diff . . . . .	22
6.1.2.8	termpos_diff . . . . .	22
6.1.2.9	timeout . . . . .	22

6.1.2.10	valueno	22
6.1.2.11	valueno_diff	22
6.1.2.12	weight	22
6.1.3	Function Documentation	23
6.1.3.1	major_version	23
6.1.3.2	minor_version	23
6.1.3.3	revision	23
6.1.3.4	sortable_serialise	23
6.1.3.5	sortable_unserialise	23
6.1.3.6	version_string	24
6.1.4	Variable Documentation	24
6.1.4.1	BAD_VALUENO	24
6.1.4.2	DB_CREATE	24
6.1.4.3	DB_CREATE_OR_OPEN	24
6.1.4.4	DB_CREATE_OR_OVERWRITE	24
6.1.4.5	DB_OPEN	24
6.2	Xapian::Auto Namespace Reference	25
6.2.1	Detailed Description	25
6.2.2	Function Documentation	25
6.2.2.1	open_stub	25
6.2.2.2	open_stub	25
6.3	Xapian::Brass Namespace Reference	26
6.3.1	Detailed Description	26
6.3.2	Function Documentation	26
6.3.2.1	open	26
6.3.2.2	open	26
6.4	Xapian::Chert Namespace Reference	28
6.4.1	Detailed Description	28
6.4.2	Function Documentation	28
6.4.2.1	open	28
6.4.2.2	open	28
6.5	Xapian::Flint Namespace Reference	30
6.5.1	Detailed Description	30
6.5.2	Function Documentation	30

6.5.2.1	open	30
6.5.2.2	open	30
6.6	Xapian::InMemory Namespace Reference	32
6.6.1	Detailed Description	32
6.6.2	Function Documentation	32
6.6.2.1	open	32
6.7	Xapian::Remote Namespace Reference	33
6.7.1	Detailed Description	33
6.7.2	Function Documentation	33
6.7.2.1	open	33
6.7.2.2	open	34
6.7.2.3	open_writable	34
6.7.2.4	open_writable	35
6.8	Xapian::Unicode Namespace Reference	36
6.8.1	Detailed Description	37
6.8.2	Enumeration Type Documentation	37
6.8.2.1	category	37
6.8.3	Function Documentation	37
6.8.3.1	nonascii_to_utf8	37
6.8.3.2	to_utf8	37
<b>7</b>	<b>Class Documentation</b>	<b>39</b>
7.1	Xapian::AssertionError Class Reference	39
7.1.1	Detailed Description	40
7.1.2	Constructor & Destructor Documentation	40
7.1.2.1	AssertionError	40
7.1.2.2	AssertionError	40
7.2	Xapian::BM25Weight Class Reference	41
7.2.1	Detailed Description	42
7.2.2	Constructor & Destructor Documentation	42
7.2.2.1	BM25Weight	42
7.2.3	Member Function Documentation	42
7.2.3.1	get_maxextra	42
7.2.3.2	get_maxpart	42

7.2.3.3	<a href="#">get_sumextra</a>	43
7.2.3.4	<a href="#">get_sumpart</a>	43
7.2.3.5	<a href="#">name</a>	43
7.2.3.6	<a href="#">serialise</a>	44
7.2.3.7	<a href="#">unserialise</a>	44
7.3	<a href="#">Xapian::BoolWeight Class Reference</a>	45
7.3.1	<a href="#">Detailed Description</a>	46
7.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	46
7.3.2.1	<a href="#">BoolWeight</a>	46
7.3.3	<a href="#">Member Function Documentation</a>	46
7.3.3.1	<a href="#">get_maxextra</a>	46
7.3.3.2	<a href="#">get_maxpart</a>	46
7.3.3.3	<a href="#">get_sumextra</a>	46
7.3.3.4	<a href="#">get_sumpart</a>	47
7.3.3.5	<a href="#">name</a>	47
7.3.3.6	<a href="#">serialise</a>	47
7.3.3.7	<a href="#">unserialise</a>	47
7.4	<a href="#">Xapian::Compactor Class Reference</a>	49
7.4.1	<a href="#">Detailed Description</a>	49
7.4.2	<a href="#">Member Function Documentation</a>	49
7.4.2.1	<a href="#">add_source</a>	49
7.4.2.2	<a href="#">resolve_duplicate_metadata</a>	50
7.4.2.3	<a href="#">set_block_size</a>	50
7.4.2.4	<a href="#">set_compaction_level</a>	50
7.4.2.5	<a href="#">set_destdir</a>	50
7.4.2.6	<a href="#">set_multipass</a>	50
7.4.2.7	<a href="#">set_renumber</a>	51
7.4.2.8	<a href="#">set_status</a>	51
7.5	<a href="#">Xapian::Database Class Reference</a>	52
7.5.1	<a href="#">Detailed Description</a>	55
7.5.2	<a href="#">Constructor &amp; Destructor Documentation</a>	56
7.5.2.1	<a href="#">Database</a>	56
7.5.2.2	<a href="#">~Database</a>	56
7.5.2.3	<a href="#">Database</a>	56

7.5.3	Member Function Documentation	56
7.5.3.1	add_database	56
7.5.3.2	allterms_begin	56
7.5.3.3	close	57
7.5.3.4	get_collection_freq	57
7.5.3.5	get_doclength_lower_bound	57
7.5.3.6	get_document	57
7.5.3.7	get_metadata	58
7.5.3.8	get_spelling_suggestion	58
7.5.3.9	get_uuid	59
7.5.3.10	get_value_freq	59
7.5.3.11	get_value_lower_bound	59
7.5.3.12	get_value_upper_bound	59
7.5.3.13	keep_alive	60
7.5.3.14	metadata_keys_begin	60
7.5.3.15	operator=	60
7.5.3.16	postlist_begin	60
7.5.3.17	reopen	61
7.5.3.18	spellings_begin	61
7.5.3.19	synonym_keys_begin	61
7.5.3.20	synonyms_begin	61
7.5.3.21	term_exists	61
7.6	Xapian::DatabaseCorruptError Class Reference	62
7.6.1	Detailed Description	62
7.6.2	Constructor & Destructor Documentation	63
7.6.2.1	DatabaseCorruptError	63
7.6.2.2	DatabaseCorruptError	63
7.7	Xapian::DatabaseCreateError Class Reference	64
7.7.1	Detailed Description	64
7.7.2	Constructor & Destructor Documentation	65
7.7.2.1	DatabaseCreateError	65
7.7.2.2	DatabaseCreateError	65
7.8	Xapian::DatabaseError Class Reference	66
7.8.1	Detailed Description	66

7.8.2	Constructor & Destructor Documentation	66
7.8.2.1	DatabaseError	66
7.8.2.2	DatabaseError	66
7.9	Xapian::DatabaseLockError Class Reference	67
7.9.1	Detailed Description	67
7.9.2	Constructor & Destructor Documentation	68
7.9.2.1	DatabaseLockError	68
7.9.2.2	DatabaseLockError	68
7.10	Xapian::DatabaseModifiedError Class Reference	69
7.10.1	Detailed Description	69
7.10.2	Constructor & Destructor Documentation	70
7.10.2.1	DatabaseModifiedError	70
7.10.2.2	DatabaseModifiedError	70
7.11	Xapian::DatabaseOpeningError Class Reference	71
7.11.1	Detailed Description	71
7.11.2	Constructor & Destructor Documentation	72
7.11.2.1	DatabaseOpeningError	72
7.11.2.2	DatabaseOpeningError	72
7.12	Xapian::DatabaseVersionError Class Reference	73
7.12.1	Detailed Description	73
7.12.2	Constructor & Destructor Documentation	74
7.12.2.1	DatabaseVersionError	74
7.12.2.2	DatabaseVersionError	74
7.13	Xapian::DateValueRangeProcessor Class Reference	75
7.13.1	Detailed Description	75
7.13.2	Constructor & Destructor Documentation	76
7.13.2.1	DateValueRangeProcessor	76
7.13.2.2	DateValueRangeProcessor	76
7.13.3	Member Function Documentation	77
7.13.3.1	operator()	77
7.14	Xapian::DecreasingValueWeightPostingSource Class Reference	78
7.14.1	Detailed Description	79
7.14.2	Member Function Documentation	80
7.14.2.1	check	80



7.14.2.2	clone	80
7.14.2.3	get_description	81
7.14.2.4	get_weight	81
7.14.2.5	init	81
7.14.2.6	name	82
7.14.2.7	next	82
7.14.2.8	serialise	82
7.14.2.9	skip_to	83
7.14.2.10	unserialise	83
7.15	Xapian::DocNotFoundError Class Reference	84
7.15.1	Detailed Description	84
7.15.2	Constructor & Destructor Documentation	84
7.15.2.1	DocNotFoundError	84
7.15.2.2	DocNotFoundError	85
7.16	Xapian::Document Class Reference	86
7.16.1	Detailed Description	88
7.16.2	Constructor & Destructor Documentation	88
7.16.2.1	Document	88
7.16.3	Member Function Documentation	88
7.16.3.1	add_boolean_term	88
7.16.3.2	add_posting	88
7.16.3.3	add_term	89
7.16.3.4	add_value	89
7.16.3.5	get_data	89
7.16.3.6	get_docid	89
7.16.3.7	get_value	90
7.16.3.8	operator=	90
7.16.3.9	remove_posting	90
7.16.3.10	remove_term	90
7.16.3.11	serialise	91
7.16.3.12	termlist_count	91
7.17	Xapian::Enquire Class Reference	92
7.17.1	Detailed Description	94
7.17.2	Constructor & Destructor Documentation	94

7.17.2.1 Enquire . . . . .	94
7.17.3 Member Function Documentation . . . . .	95
7.17.3.1 add_matchspy . . . . .	95
7.17.3.2 get_eset . . . . .	95
7.17.3.3 get_eset . . . . .	96
7.17.3.4 get_eset . . . . .	96
7.17.3.5 get_matching_terms_begin . . . . .	97
7.17.3.6 get_matching_terms_begin . . . . .	97
7.17.3.7 get_mset . . . . .	98
7.17.3.8 get_mset . . . . .	99
7.17.3.9 get_mset . . . . .	100
7.17.3.10 get_query . . . . .	101
7.17.3.11 set_collapse_key . . . . .	101
7.17.3.12 set_cutoff . . . . .	102
7.17.3.13 set_docid_order . . . . .	102
7.17.3.14 set_query . . . . .	103
7.17.3.15 set_sort_by_key . . . . .	103
7.17.3.16 set_sort_by_key_then_relevance . . . . .	103
7.17.3.17 set_sort_by_relevance . . . . .	103
7.17.3.18 set_sort_by_relevance_then_key . . . . .	103
7.17.3.19 set_sort_by_relevance_then_value . . . . .	104
7.17.3.20 set_sort_by_value . . . . .	104
7.17.3.21 set_sort_by_value_then_relevance . . . . .	104
7.17.3.22 set_weighting_scheme . . . . .	105
7.18 Xapian::Error Class Reference . . . . .	106
7.18.1 Detailed Description . . . . .	106
7.18.2 Member Function Documentation . . . . .	107
7.18.2.1 get_context . . . . .	107
7.18.2.2 get_error_string . . . . .	107
7.19 Xapian::ErrorHandler Class Reference . . . . .	108
7.19.1 Detailed Description . . . . .	108
7.19.2 Member Function Documentation . . . . .	108
7.19.2.1 operator() . . . . .	108
7.20 Xapian::ESet Class Reference . . . . .	109

7.20.1	Detailed Description	110
7.20.2	Member Function Documentation	110
7.20.2.1	get_ebound	110
7.20.2.2	max_size	110
7.20.2.3	operator[]	110
7.21	Xapian::ESetIterator Class Reference	111
7.21.1	Detailed Description	112
7.22	Xapian::ExpandDecider Class Reference	113
7.22.1	Detailed Description	113
7.22.2	Constructor & Destructor Documentation	113
7.22.2.1	~ExpandDecider	113
7.23	Xapian::ExpandDeciderAnd Class Reference	114
7.23.1	Detailed Description	114
7.23.2	Constructor & Destructor Documentation	114
7.23.2.1	ExpandDeciderAnd	114
7.24	Xapian::ExpandDeciderFilterTerms Class Reference	115
7.24.1	Detailed Description	115
7.24.2	Constructor & Destructor Documentation	115
7.24.2.1	ExpandDeciderFilterTerms	115
7.25	Xapian::FeatureUnavailableError Class Reference	117
7.25.1	Detailed Description	117
7.25.2	Constructor & Destructor Documentation	117
7.25.2.1	FeatureUnavailableError	117
7.25.2.2	FeatureUnavailableError	118
7.26	Xapian::FixedWeightPostingSource Class Reference	119
7.26.1	Detailed Description	120
7.26.2	Constructor & Destructor Documentation	120
7.26.2.1	FixedWeightPostingSource	120
7.26.3	Member Function Documentation	120
7.26.3.1	at_end	120
7.26.3.2	check	121
7.26.3.3	clone	121
7.26.3.4	get_description	122
7.26.3.5	get_docid	122

7.26.3.6	<a href="#">get_termfreq_est</a>	122
7.26.3.7	<a href="#">get_termfreq_max</a>	122
7.26.3.8	<a href="#">get_termfreq_min</a>	123
7.26.3.9	<a href="#">get_weight</a>	123
7.26.3.10	<a href="#">init</a>	123
7.26.3.11	<a href="#">name</a>	124
7.26.3.12	<a href="#">next</a>	124
7.26.3.13	<a href="#">serialise</a>	124
7.26.3.14	<a href="#">skip_to</a>	125
7.26.3.15	<a href="#">unserialise</a>	125
7.27	<a href="#">Xapian::InternalError Class Reference</a>	126
7.27.1	<a href="#">Detailed Description</a>	126
7.27.2	<a href="#">Constructor &amp; Destructor Documentation</a>	126
7.27.2.1	<a href="#">InternalError</a>	126
7.27.2.2	<a href="#">InternalError</a>	127
7.28	<a href="#">Xapian::InvalidArgumentError Class Reference</a>	128
7.28.1	<a href="#">Detailed Description</a>	128
7.28.2	<a href="#">Constructor &amp; Destructor Documentation</a>	128
7.28.2.1	<a href="#">InvalidArgumentError</a>	128
7.28.2.2	<a href="#">InvalidArgumentError</a>	129
7.29	<a href="#">Xapian::InvalidOperationError Class Reference</a>	130
7.29.1	<a href="#">Detailed Description</a>	130
7.29.2	<a href="#">Constructor &amp; Destructor Documentation</a>	130
7.29.2.1	<a href="#">InvalidOperationError</a>	130
7.29.2.2	<a href="#">InvalidOperationError</a>	131
7.30	<a href="#">Xapian::KeyMaker Class Reference</a>	132
7.30.1	<a href="#">Detailed Description</a>	132
7.30.2	<a href="#">Constructor &amp; Destructor Documentation</a>	132
7.30.2.1	<a href="#">~KeyMaker</a>	132
7.30.3	<a href="#">Member Function Documentation</a>	132
7.30.3.1	<a href="#">operator()</a>	132
7.31	<a href="#">Xapian::LogicError Class Reference</a>	134
7.31.1	<a href="#">Detailed Description</a>	134
7.32	<a href="#">Xapian::MatchDecider Class Reference</a>	135

7.32.1 Detailed Description . . . . .	135
7.32.2 Member Function Documentation . . . . .	135
7.32.2.1 operator() . . . . .	135
7.33 Xapian::MatchSpy Class Reference . . . . .	136
7.33.1 Detailed Description . . . . .	137
7.33.2 Constructor & Destructor Documentation . . . . .	137
7.33.2.1 ~MatchSpy . . . . .	137
7.33.3 Member Function Documentation . . . . .	137
7.33.3.1 clone . . . . .	137
7.33.3.2 get_description . . . . .	137
7.33.3.3 merge_results . . . . .	138
7.33.3.4 name . . . . .	138
7.33.3.5 operator() . . . . .	138
7.33.3.6 serialise . . . . .	138
7.33.3.7 serialise_results . . . . .	139
7.33.3.8 unserialise . . . . .	139
7.34 Xapian::MSet Class Reference . . . . .	140
7.34.1 Detailed Description . . . . .	142
7.34.2 Member Function Documentation . . . . .	142
7.34.2.1 convert_to_percent . . . . .	142
7.34.2.2 fetch . . . . .	143
7.34.2.3 get_firstitem . . . . .	143
7.34.2.4 get_matches_estimated . . . . .	143
7.34.2.5 get_matches_lower_bound . . . . .	143
7.34.2.6 get_matches_upper_bound . . . . .	143
7.34.2.7 get_max_attained . . . . .	144
7.34.2.8 get_max_possible . . . . .	144
7.34.2.9 get_termfreq . . . . .	144
7.34.2.10 get_termweight . . . . .	144
7.34.2.11 max_size . . . . .	145
7.34.2.12 operator[] . . . . .	145
7.35 Xapian::MSetIterator Class Reference . . . . .	146
7.35.1 Detailed Description . . . . .	147
7.35.2 Member Function Documentation . . . . .	147

7.35.2.1	<a href="#">get_collapse_count</a>	147
7.35.2.2	<a href="#">get_document</a>	148
7.35.2.3	<a href="#">get_percent</a>	148
7.35.2.4	<a href="#">get_rank</a>	148
7.36	<a href="#">Xapian::MultiValueKeyMaker Class Reference</a>	150
7.36.1	<a href="#">Detailed Description</a>	150
7.36.2	<a href="#">Member Function Documentation</a>	150
7.36.2.1	<a href="#">operator()</a>	150
7.37	<a href="#">Xapian::MultiValueSorter Class Reference</a>	152
7.37.1	<a href="#">Detailed Description</a>	152
7.37.2	<a href="#">Member Function Documentation</a>	153
7.37.2.1	<a href="#">operator()</a>	153
7.38	<a href="#">Xapian::NetworkError Class Reference</a>	154
7.38.1	<a href="#">Detailed Description</a>	154
7.38.2	<a href="#">Constructor &amp; Destructor Documentation</a>	155
7.38.2.1	<a href="#">NetworkError</a>	155
7.38.2.2	<a href="#">NetworkError</a>	155
7.39	<a href="#">Xapian::NetworkTimeoutError Class Reference</a>	156
7.39.1	<a href="#">Detailed Description</a>	156
7.39.2	<a href="#">Constructor &amp; Destructor Documentation</a>	157
7.39.2.1	<a href="#">NetworkTimeoutError</a>	157
7.39.2.2	<a href="#">NetworkTimeoutError</a>	157
7.40	<a href="#">Xapian::NumberValueRangeProcessor Class Reference</a>	158
7.40.1	<a href="#">Detailed Description</a>	158
7.40.2	<a href="#">Constructor &amp; Destructor Documentation</a>	159
7.40.2.1	<a href="#">NumberValueRangeProcessor</a>	159
7.40.2.2	<a href="#">NumberValueRangeProcessor</a>	159
7.40.3	<a href="#">Member Function Documentation</a>	159
7.40.3.1	<a href="#">operator()</a>	159
7.41	<a href="#">Xapian::PositionIterator Class Reference</a>	161
7.41.1	<a href="#">Detailed Description</a>	161
7.41.2	<a href="#">Constructor &amp; Destructor Documentation</a>	161
7.41.2.1	<a href="#">PositionIterator</a>	161
7.41.3	<a href="#">Member Function Documentation</a>	162

7.41.3.1	operator=	162
7.41.3.2	skip_to	162
7.42	Xapian::PostingIterator Class Reference	163
7.42.1	Detailed Description	164
7.42.2	Constructor & Destructor Documentation	164
7.42.2.1	PostingIterator	164
7.42.3	Member Function Documentation	164
7.42.3.1	get_doclength	164
7.42.3.2	operator=	164
7.42.3.3	skip_to	165
7.43	Xapian::PostingSource Class Reference	166
7.43.1	Detailed Description	167
7.43.2	Member Function Documentation	167
7.43.2.1	at_end	167
7.43.2.2	check	168
7.43.2.3	clone	168
7.43.2.4	get_description	169
7.43.2.5	get_docid	169
7.43.2.6	get_termfreq_est	169
7.43.2.7	get_termfreq_max	169
7.43.2.8	get_termfreq_min	170
7.43.2.9	get_weight	170
7.43.2.10	init	170
7.43.2.11	name	171
7.43.2.12	next	171
7.43.2.13	serialise	172
7.43.2.14	set_maxweight	172
7.43.2.15	skip_to	172
7.43.2.16	unserialise	173
7.44	Xapian::Query Class Reference	174
7.44.1	Detailed Description	176
7.44.2	Member Enumeration Documentation	176
7.44.2.1	op	176
7.44.3	Constructor & Destructor Documentation	177

7.44.3.1	Query	177
7.44.3.2	Query	177
7.44.3.3	~Query	177
7.44.3.4	Query	177
7.44.3.5	Query	177
7.44.3.6	Query	177
7.44.3.7	Query	178
7.44.3.8	Query	178
7.44.3.9	Query	178
7.44.3.10	Query	179
7.44.4	Member Function Documentation	179
7.44.4.1	empty	179
7.44.4.2	get_length	179
7.44.4.3	get_terms_begin	179
7.44.4.4	operator=	179
7.44.4.5	serialise	179
7.44.4.6	unserialise	180
7.44.4.7	unserialise	180
7.44.5	Member Data Documentation	180
7.44.5.1	MatchAll	180
7.44.5.2	MatchNothing	180
7.45	Xapian::QueryParser Class Reference	181
7.45.1	Detailed Description	182
7.45.2	Member Enumeration Documentation	182
7.45.2.1	feature_flag	182
7.45.3	Member Function Documentation	184
7.45.3.1	add_boolean_prefix	184
7.45.3.2	add_prefix	185
7.45.3.3	get_corrected_query_string	185
7.45.3.4	get_default_op	185
7.45.3.5	parse_query	186
7.45.3.6	set_database	186
7.45.3.7	set_default_op	186
7.45.3.8	set_max_wildcard_expansion	187



7.45.3.9	<a href="#">set_stemmer</a>	187
7.45.3.10	<a href="#">set_stemming_strategy</a>	187
7.46	<a href="#">Xapian::QueryParserError Class Reference</a>	189
7.46.1	<a href="#">Detailed Description</a>	189
7.46.2	<a href="#">Constructor &amp; Destructor Documentation</a>	189
7.46.2.1	<a href="#">QueryParserError</a>	189
7.46.2.2	<a href="#">QueryParserError</a>	190
7.47	<a href="#">Xapian::RangeError Class Reference</a>	191
7.47.1	<a href="#">Detailed Description</a>	191
7.47.2	<a href="#">Constructor &amp; Destructor Documentation</a>	191
7.47.2.1	<a href="#">RangeError</a>	191
7.47.2.2	<a href="#">RangeError</a>	192
7.48	<a href="#">Xapian::Registry Class Reference</a>	193
7.48.1	<a href="#">Detailed Description</a>	193
7.48.2	<a href="#">Constructor &amp; Destructor Documentation</a>	194
7.48.2.1	<a href="#">Registry</a>	194
7.48.2.2	<a href="#">Registry</a>	194
7.48.3	<a href="#">Member Function Documentation</a>	194
7.48.3.1	<a href="#">get_match_spy</a>	194
7.48.3.2	<a href="#">get_posting_source</a>	194
7.48.3.3	<a href="#">get_weighting_scheme</a>	194
7.48.3.4	<a href="#">operator=</a>	194
7.49	<a href="#">Xapian::RSet Class Reference</a>	196
7.49.1	<a href="#">Detailed Description</a>	197
7.50	<a href="#">Xapian::RuntimeError Class Reference</a>	198
7.50.1	<a href="#">Detailed Description</a>	198
7.51	<a href="#">Xapian::SerialisationError Class Reference</a>	199
7.51.1	<a href="#">Detailed Description</a>	199
7.51.2	<a href="#">Constructor &amp; Destructor Documentation</a>	199
7.51.2.1	<a href="#">SerialisationError</a>	199
7.51.2.2	<a href="#">SerialisationError</a>	200
7.52	<a href="#">Xapian::SimpleStopper Class Reference</a>	201
7.52.1	<a href="#">Detailed Description</a>	201
7.53	<a href="#">Xapian::Sorter Class Reference</a>	202

7.53.1 Detailed Description . . . . .	202
7.54 Xapian::Stem Class Reference . . . . .	203
7.54.1 Detailed Description . . . . .	203
7.54.2 Constructor & Destructor Documentation . . . . .	204
7.54.2.1 Stem . . . . .	204
7.54.2.2 Stem . . . . .	204
7.54.2.3 Stem . . . . .	205
7.54.3 Member Function Documentation . . . . .	205
7.54.3.1 get_available_languages . . . . .	205
7.54.3.2 operator() . . . . .	205
7.55 Xapian::StemImplementation Struct Reference . . . . .	206
7.55.1 Detailed Description . . . . .	206
7.56 Xapian::Stopper Class Reference . . . . .	207
7.56.1 Detailed Description . . . . .	207
7.57 Xapian::StringValueRangeProcessor Class Reference . . . . .	208
7.57.1 Detailed Description . . . . .	208
7.57.2 Constructor & Destructor Documentation . . . . .	208
7.57.2.1 StringValueRangeProcessor . . . . .	208
7.57.2.2 StringValueRangeProcessor . . . . .	209
7.57.3 Member Function Documentation . . . . .	209
7.57.3.1 operator() . . . . .	209
7.58 Xapian::TermGenerator Class Reference . . . . .	210
7.58.1 Detailed Description . . . . .	211
7.58.2 Member Enumeration Documentation . . . . .	211
7.58.2.1 flags . . . . .	211
7.58.3 Member Function Documentation . . . . .	212
7.58.3.1 increase_termpos . . . . .	212
7.58.3.2 index_text . . . . .	212
7.58.3.3 index_text . . . . .	212
7.58.3.4 index_text_without_positions . . . . .	212
7.58.3.5 index_text_without_positions . . . . .	212
7.58.3.6 set_flags . . . . .	213
7.58.3.7 set_stopper . . . . .	213
7.59 Xapian::TermIterator Class Reference . . . . .	214

7.59.1	Detailed Description	215
7.59.2	Constructor & Destructor Documentation	215
7.59.2.1	TermIterator	215
7.59.3	Member Function Documentation	215
7.59.3.1	get_termfreq	215
7.59.3.2	get_wdf	215
7.59.3.3	operator=	215
7.59.3.4	skip_to	216
7.60	Xapian::TradWeight Class Reference	217
7.60.1	Detailed Description	218
7.60.2	Constructor & Destructor Documentation	218
7.60.2.1	TradWeight	218
7.60.3	Member Function Documentation	218
7.60.3.1	get_maxextra	218
7.60.3.2	get_maxpart	218
7.60.3.3	get_sumextra	218
7.60.3.4	get_sumpart	219
7.60.3.5	name	219
7.60.3.6	serialise	219
7.60.3.7	unserialise	220
7.61	Xapian::UnimplementedError Class Reference	221
7.61.1	Detailed Description	221
7.61.2	Constructor & Destructor Documentation	221
7.61.2.1	UnimplementedError	221
7.61.2.2	UnimplementedError	222
7.62	Xapian::Utf8Iterator Class Reference	223
7.62.1	Detailed Description	224
7.62.2	Constructor & Destructor Documentation	224
7.62.2.1	Utf8Iterator	224
7.62.2.2	Utf8Iterator	224
7.62.2.3	Utf8Iterator	225
7.62.2.4	Utf8Iterator	225
7.62.3	Member Function Documentation	225
7.62.3.1	assign	225

7.62.3.2	assign	225
7.62.3.3	left	226
7.62.3.4	operator!=	226
7.62.3.5	operator*	226
7.62.3.6	operator++	226
7.62.3.7	operator++	226
7.62.3.8	operator==	226
7.62.3.9	raw	227
7.63	Xapian::ValueCollectionMatchSpy Class Reference	228
7.63.1	Detailed Description	229
7.63.2	Member Function Documentation	229
7.63.2.1	clone	229
7.63.2.2	get_description	229
7.63.2.3	get_total	230
7.63.2.4	merge_results	230
7.63.2.5	name	230
7.63.2.6	operator()	230
7.63.2.7	serialise	230
7.63.2.8	serialise_results	231
7.63.2.9	top_values_begin	231
7.63.2.10	unserialise	231
7.63.2.11	values_begin	231
7.64	Xapian::ValueIterator Class Reference	233
7.64.1	Detailed Description	233
7.64.2	Constructor & Destructor Documentation	234
7.64.2.1	ValueIterator	234
7.64.3	Member Function Documentation	234
7.64.3.1	check	234
7.64.3.2	get_docid	234
7.64.3.3	get_valueno	234
7.64.3.4	skip_to	234
7.65	Xapian::ValueMapPostingSource Class Reference	236
7.65.1	Detailed Description	237
7.65.2	Constructor & Destructor Documentation	237

7.65.2.1	ValueMapPostingSource . . . . .	237
7.65.3	Member Function Documentation . . . . .	237
7.65.3.1	add_mapping . . . . .	237
7.65.3.2	clear_mappings . . . . .	237
7.65.3.3	clone . . . . .	238
7.65.3.4	get_description . . . . .	238
7.65.3.5	get_weight . . . . .	238
7.65.3.6	init . . . . .	238
7.65.3.7	name . . . . .	239
7.65.3.8	serialise . . . . .	239
7.65.3.9	set_default_weight . . . . .	240
7.65.3.10	unserialise . . . . .	240
7.66	Xapian::ValuePostingSource Class Reference . . . . .	241
7.66.1	Detailed Description . . . . .	242
7.66.2	Constructor & Destructor Documentation . . . . .	243
7.66.2.1	ValuePostingSource . . . . .	243
7.66.3	Member Function Documentation . . . . .	243
7.66.3.1	at_end . . . . .	243
7.66.3.2	check . . . . .	243
7.66.3.3	get_docid . . . . .	244
7.66.3.4	get_termfreq_est . . . . .	244
7.66.3.5	get_termfreq_max . . . . .	244
7.66.3.6	get_termfreq_min . . . . .	244
7.66.3.7	init . . . . .	244
7.66.3.8	next . . . . .	245
7.66.3.9	skip_to . . . . .	245
7.66.4	Member Data Documentation . . . . .	246
7.66.4.1	termfreq_est . . . . .	246
7.66.4.2	termfreq_max . . . . .	246
7.66.4.3	termfreq_min . . . . .	246
7.67	Xapian::ValueRangeProcessor Struct Reference . . . . .	247
7.67.1	Detailed Description . . . . .	247
7.67.2	Member Function Documentation . . . . .	247
7.67.2.1	operator() . . . . .	247

7.68	Xapian::ValueSetMatchDecider Class Reference . . . . .	248
7.68.1	Detailed Description . . . . .	248
7.68.2	Constructor & Destructor Documentation . . . . .	248
7.68.2.1	ValueSetMatchDecider . . . . .	248
7.68.3	Member Function Documentation . . . . .	249
7.68.3.1	add_value . . . . .	249
7.68.3.2	operator() . . . . .	249
7.68.3.3	remove_value . . . . .	249
7.69	Xapian::ValueWeightPostingSource Class Reference . . . . .	250
7.69.1	Detailed Description . . . . .	251
7.69.2	Constructor & Destructor Documentation . . . . .	251
7.69.2.1	ValueWeightPostingSource . . . . .	251
7.69.3	Member Function Documentation . . . . .	251
7.69.3.1	clone . . . . .	251
7.69.3.2	get_description . . . . .	252
7.69.3.3	get_weight . . . . .	252
7.69.3.4	init . . . . .	252
7.69.3.5	name . . . . .	253
7.69.3.6	serialise . . . . .	253
7.69.3.7	unserialise . . . . .	254
7.70	Xapian::Weight Class Reference . . . . .	255
7.70.1	Detailed Description . . . . .	257
7.70.2	Constructor & Destructor Documentation . . . . .	257
7.70.2.1	~Weight . . . . .	257
7.70.2.2	Weight . . . . .	257
7.70.3	Member Function Documentation . . . . .	257
7.70.3.1	clone . . . . .	257
7.70.3.2	get_doclength_lower_bound . . . . .	257
7.70.3.3	get_doclength_upper_bound . . . . .	258
7.70.3.4	get_maxextra . . . . .	258
7.70.3.5	get_maxpart . . . . .	258
7.70.3.6	get_sumextra . . . . .	258
7.70.3.7	get_sumpart . . . . .	258
7.70.3.8	get_wdf_upper_bound . . . . .	259

7.70.3.9	init	259
7.70.3.10	name	259
7.70.3.11	need_stat	259
7.70.3.12	serialise	260
7.70.3.13	unserialise	260
7.71	Xapian::WritableDatabase Class Reference	261
7.71.1	Detailed Description	263
7.71.2	Constructor & Destructor Documentation	263
7.71.2.1	~WritableDatabase	263
7.71.2.2	WritableDatabase	263
7.71.2.3	WritableDatabase	263
7.71.3	Member Function Documentation	264
7.71.3.1	add_document	264
7.71.3.2	add_spelling	264
7.71.3.3	add_synonym	264
7.71.3.4	begin_transaction	265
7.71.3.5	cancel_transaction	265
7.71.3.6	clear_synonyms	266
7.71.3.7	commit	266
7.71.3.8	commit_transaction	266
7.71.3.9	delete_document	267
7.71.3.10	delete_document	267
7.71.3.11	flush	268
7.71.3.12	operator=	268
7.71.3.13	remove_spelling	268
7.71.3.14	remove_synonym	269
7.71.3.15	replace_document	269
7.71.3.16	replace_document	270
7.71.3.17	set_metadata	270
<b>8</b>	<b>File Documentation</b>	<b>273</b>
8.1	xapian/error.h File Reference	273
8.1.1	Detailed Description	275
8.2	xapian/version.h File Reference	276

8.2.1	Detailed Description . . . . .	276
8.2.2	Define Documentation . . . . .	277
8.2.2.1	XAPIAN_MAJOR_VERSION . . . . .	277
8.2.2.2	XAPIAN_MINOR_VERSION . . . . .	277
8.2.2.3	XAPIAN_REVISION . . . . .	277
8.3	xapian.h File Reference . . . . .	278
8.3.1	Detailed Description . . . . .	278
8.4	xapian/compactor.h File Reference . . . . .	279
8.4.1	Detailed Description . . . . .	279
8.5	xapian/database.h File Reference . . . . .	280
8.5.1	Detailed Description . . . . .	280
8.6	xapian/dbfactory.h File Reference . . . . .	281
8.6.1	Detailed Description . . . . .	282
8.7	xapian/document.h File Reference . . . . .	283
8.7.1	Detailed Description . . . . .	283
8.8	xapian/enquire.h File Reference . . . . .	284
8.8.1	Detailed Description . . . . .	285
8.9	xapian/errorhandler.h File Reference . . . . .	286
8.9.1	Detailed Description . . . . .	286
8.10	xapian/expanddecider.h File Reference . . . . .	287
8.10.1	Detailed Description . . . . .	287
8.11	xapian/keymaker.h File Reference . . . . .	288
8.11.1	Detailed Description . . . . .	288
8.12	xapian/matchspy.h File Reference . . . . .	289
8.12.1	Detailed Description . . . . .	289
8.13	xapian/positioniterator.h File Reference . . . . .	290
8.13.1	Detailed Description . . . . .	290
8.14	xapian/postingiterator.h File Reference . . . . .	291
8.14.1	Detailed Description . . . . .	291
8.15	xapian/postingsource.h File Reference . . . . .	292
8.15.1	Detailed Description . . . . .	292
8.16	xapian/query.h File Reference . . . . .	293
8.16.1	Detailed Description . . . . .	293
8.17	xapian/queryparser.h File Reference . . . . .	294



---

8.17.1 Detailed Description . . . . .	295
8.18 xapian/registry.h File Reference . . . . .	296
8.18.1 Detailed Description . . . . .	296
8.19 xapian/stem.h File Reference . . . . .	297
8.19.1 Detailed Description . . . . .	297
8.20 xapian/termgenerator.h File Reference . . . . .	298
8.20.1 Detailed Description . . . . .	298
8.21 xapian/termiterator.h File Reference . . . . .	299
8.21.1 Detailed Description . . . . .	299
8.22 xapian/types.h File Reference . . . . .	300
8.22.1 Detailed Description . . . . .	301
8.23 xapian/unicode.h File Reference . . . . .	302
8.23.1 Detailed Description . . . . .	303
8.24 xapian/valueiterator.h File Reference . . . . .	304
8.24.1 Detailed Description . . . . .	304
8.25 xapian/valuesetmatchdecider.h File Reference . . . . .	305
8.25.1 Detailed Description . . . . .	305
8.26 xapian/weight.h File Reference . . . . .	306
8.26.1 Detailed Description . . . . .	306



# Chapter 1

## Deprecated List

**Member `Xapian::Enquire::get_mset(Xapian::doccount first, Xapian::doccount maxitems, Xapian::doccount checkat)`** this parameter is deprecated - use the newer MatchSpy class and `add_matchspy()` method instead.

**Class `Xapian::MultiValueSorter`** This class is deprecated - you should migrate to using MultiValueKeyMaker instead. Note that `MultiValueSorter::add()` becomes `MultiValueKeyMaker::add_value()`, but the sense of the direction flag is reversed (to be consistent with `Enquire::set_sort_by_value()`), so:



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">Xapian</a> (The <a href="#">Xapian</a> namespace contains public interfaces for the <a href="#">Xapian</a> library ) . . . . .	13
<a href="#">Xapian::Auto</a> ( <a href="#">Database</a> factory functions which determine the database type automatically ) . . . . .	25
<a href="#">Xapian::Brass</a> ( <a href="#">Database</a> factory functions for the brass backend ) . . . . .	26
<a href="#">Xapian::Chert</a> ( <a href="#">Database</a> factory functions for the chert backend ) . . . . .	28
<a href="#">Xapian::Flint</a> ( <a href="#">Database</a> factory functions for the flint backend ) . . . . .	30
<a href="#">Xapian::InMemory</a> ( <a href="#">Database</a> factory functions for the inmemory backend ) .	32
<a href="#">Xapian::Remote</a> ( <a href="#">Database</a> factory functions for the remote backend ) . . . .	33
<a href="#">Xapian::Unicode</a> (Functions associated with handling <a href="#">Unicode</a> characters ) .	36



## Chapter 3

# Class Index

### 3.1 Class Hierarchy

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

Xapian::Compactor . . . . .	49
Xapian::Database . . . . .	52
Xapian::WritableDatabase . . . . .	261
Xapian::Document . . . . .	86
Xapian::Enquire . . . . .	92
Xapian::Error . . . . .	106
Xapian::LogicError . . . . .	134
Xapian::AssertionError . . . . .	39
Xapian::InvalidArgumentError . . . . .	128
Xapian::InvalidOperationError . . . . .	130
Xapian::UnimplementedError . . . . .	221
Xapian::RuntimeError . . . . .	198
Xapian::DatabaseError . . . . .	66
Xapian::DatabaseCorruptError . . . . .	62
Xapian::DatabaseCreateError . . . . .	64
Xapian::DatabaseLockError . . . . .	67
Xapian::DatabaseModifiedError . . . . .	69
Xapian::DatabaseOpeningError . . . . .	71
Xapian::DatabaseVersionError . . . . .	73
Xapian::DocNotFoundError . . . . .	84
Xapian::FeatureUnavailableError . . . . .	117
Xapian::InternalError . . . . .	126
Xapian::NetworkError . . . . .	154
Xapian::NetworkTimeoutError . . . . .	156
Xapian::QueryParserError . . . . .	189
Xapian::RangeError . . . . .	191
Xapian::SerialisationError . . . . .	199
Xapian::ErrorHandler . . . . .	108

Xapian::ESet . . . . .	109
Xapian::ESetIterator . . . . .	111
Xapian::ExpandDecider . . . . .	113
Xapian::ExpandDeciderAnd . . . . .	114
Xapian::ExpandDeciderFilterTerms . . . . .	115
Xapian::KeyMaker . . . . .	132
Xapian::MultiValueKeyMaker . . . . .	150
Xapian::Sorter . . . . .	202
Xapian::MultiValueSorter . . . . .	152
Xapian::MatchDecider . . . . .	135
Xapian::ValueSetMatchDecider . . . . .	248
Xapian::MatchSpy . . . . .	136
Xapian::ValueCountMatchSpy . . . . .	228
Xapian::MSet . . . . .	140
Xapian::MSetIterator . . . . .	146
Xapian::PositionIterator . . . . .	161
Xapian::PostingIterator . . . . .	163
Xapian::PostingSource . . . . .	166
Xapian::FixedWeightPostingSource . . . . .	119
Xapian::ValuePostingSource . . . . .	241
Xapian::ValueMapPostingSource . . . . .	236
Xapian::ValueWeightPostingSource . . . . .	250
Xapian::DecreasingValueWeightPostingSource . . . . .	78
Xapian::Query . . . . .	174
Xapian::QueryParser . . . . .	181
Xapian::Registry . . . . .	193
Xapian::RSet . . . . .	196
Xapian::Stem . . . . .	203
Xapian::StemImplementation . . . . .	206
Xapian::Stopper . . . . .	207
Xapian::SimpleStopper . . . . .	201
Xapian::TermGenerator . . . . .	210
Xapian::TermIterator . . . . .	214
Xapian::Utf8Iterator . . . . .	223
Xapian::ValueIterator . . . . .	233
Xapian::ValueRangeProcessor . . . . .	247
Xapian::StringValueRangeProcessor . . . . .	208
Xapian::DateValueRangeProcessor . . . . .	75
Xapian::NumberValueRangeProcessor . . . . .	158
Xapian::Weight . . . . .	255
Xapian::BM25Weight . . . . .	41
Xapian::BoolWeight . . . . .	45
Xapian::TradWeight . . . . .	217



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">Xapian::AssertionError</a> ( <a href="#">AssertionError</a> is thrown if a logical assertion inside <a href="#">Xapian</a> fails ) . . . . .	39
<a href="#">Xapian::BM25Weight</a> ( <a href="#">Xapian::Weight</a> subclass implementing the BM25 probabilistic formula ) . . . . .	41
<a href="#">Xapian::BoolWeight</a> (Class implementing a "boolean" weighting scheme ) . .	45
<a href="#">Xapian::Compactor</a> (Compact a database, or merge and compact several ) . .	49
<a href="#">Xapian::Database</a> (This class is used to access a database, or a group of databases ) . . . . .	52
<a href="#">Xapian::DatabaseCorruptError</a> ( <a href="#">DatabaseCorruptError</a> indicates database corruption was detected ) . . . . .	62
<a href="#">Xapian::DatabaseCreateError</a> ( <a href="#">DatabaseCreateError</a> indicates a failure to create a database ) . . . . .	64
<a href="#">Xapian::DatabaseError</a> ( <a href="#">DatabaseError</a> indicates some sort of database related error ) . . . . .	66
<a href="#">Xapian::DatabaseLockError</a> ( <a href="#">DatabaseLockError</a> indicates failure to lock a database ) . . . . .	67
<a href="#">Xapian::DatabaseModifiedError</a> ( <a href="#">DatabaseModifiedError</a> indicates a database was modified ) . . . . .	69
<a href="#">Xapian::DatabaseOpeningError</a> ( <a href="#">DatabaseOpeningError</a> indicates failure to open a database ) . . . . .	71
<a href="#">Xapian::DatabaseVersionError</a> ( <a href="#">DatabaseVersionError</a> indicates that a database is in an unsupported format ) . . . . .	73
<a href="#">Xapian::DateValueRangeProcessor</a> (Handle a date range ) . . . . .	75
<a href="#">Xapian::DecreasingValueWeightPostingSource</a> (Read weights from a value which is known to decrease as docid increases ) . . . . .	78
<a href="#">Xapian::DocNotFoundError</a> (Indicates an attempt to access a document not present in the database ) . . . . .	84
<a href="#">Xapian::Document</a> (A handle representing a document in a <a href="#">Xapian</a> database )	86

<a href="#">Xapian::Enquire</a> (This class provides an interface to the information retrieval system for the purpose of searching ) . . . . .	92
<a href="#">Xapian::Error</a> (All exceptions thrown by <a href="#">Xapian</a> are subclasses of <a href="#">Xapian::Error</a> ) . . . . .	106
<a href="#">Xapian::ErrorHandler</a> (Decide if a <a href="#">Xapian::Error</a> exception should be ignored ) . . . . .	108
<a href="#">Xapian::ESet</a> (Class representing an ordered set of expand terms (an <a href="#">ESet</a> ) ) . . . . .	109
<a href="#">Xapian::ESetIterator</a> (Iterate through terms in the <a href="#">ESet</a> ) . . . . .	111
<a href="#">Xapian::ExpandDecider</a> (Virtual base class for expand decider functor ) . . . . .	113
<a href="#">Xapian::ExpandDeciderAnd</a> ( <a href="#">ExpandDecider</a> subclass which rejects terms using two <a href="#">ExpandDeciders</a> ) . . . . .	114
<a href="#">Xapian::ExpandDeciderFilterTerms</a> ( <a href="#">ExpandDecider</a> subclass which rejects terms in a specified list ) . . . . .	115
<a href="#">Xapian::FeatureUnavailableError</a> (Indicates an attempt to use a feature which is unavailable ) . . . . .	117
<a href="#">Xapian::FixedWeightPostingSource</a> (A posting source which returns a fixed weight for all documents ) . . . . .	119
<a href="#">Xapian::InternalError</a> ( <a href="#">InternalError</a> indicates a runtime problem of some sort ) . . . . .	126
<a href="#">Xapian::InvalidArgumentError</a> ( <a href="#">InvalidArgumentError</a> indicates an invalid parameter value was passed to the API ) . . . . .	128
<a href="#">Xapian::InvalidOperationError</a> ( <a href="#">InvalidOperationError</a> indicates the API was used in an invalid way ) . . . . .	130
<a href="#">Xapian::KeyMaker</a> (Virtual base class for key making functors ) . . . . .	132
<a href="#">Xapian::LogicError</a> (The base class for exceptions indicating errors in the program logic ) . . . . .	134
<a href="#">Xapian::MatchDecider</a> (Base class for matcher decision functor ) . . . . .	135
<a href="#">Xapian::MatchSpy</a> (Abstract base class for match spies ) . . . . .	136
<a href="#">Xapian::MSet</a> (A match set ( <a href="#">MSet</a> ) ) . . . . .	140
<a href="#">Xapian::MSetIterator</a> (An iterator pointing to items in an <a href="#">MSet</a> ) . . . . .	146
<a href="#">Xapian::MultiValueKeyMaker</a> ( <a href="#">KeyMaker</a> subclass which combines several values ) . . . . .	150
<a href="#">Xapian::MultiValueSorter</a> ( <a href="#">Sorter</a> subclass which sorts by a several values ) . . . . .	152
<a href="#">Xapian::NetworkError</a> (Indicates a problem communicating with a remote database ) . . . . .	154
<a href="#">Xapian::NetworkTimeoutError</a> (Indicates a timeout expired while communicating with a remote database ) . . . . .	156
<a href="#">Xapian::NumberValueRangeProcessor</a> (Handle a number range ) . . . . .	158
<a href="#">Xapian::PositionIterator</a> (An iterator pointing to items in a list of positions ) . . . . .	161
<a href="#">Xapian::PostingIterator</a> (An iterator pointing to items in a list of postings ) . . . . .	163
<a href="#">Xapian::PostingSource</a> (Base class which provides an "external" source of postings ) . . . . .	166
<a href="#">Xapian::Query</a> (Class representing a query ) . . . . .	174
<a href="#">Xapian::QueryParser</a> (Build a <a href="#">Xapian::Query</a> object from a user query string ) . . . . .	181
<a href="#">Xapian::QueryParserError</a> (Indicates a query string can't be parsed ) . . . . .	189
<a href="#">Xapian::RangeError</a> ( <a href="#">RangeError</a> indicates an attempt to access outside the bounds of a container ) . . . . .	191
<a href="#">Xapian::Registry</a> ( <a href="#">Registry</a> for user subclasses ) . . . . .	193
<a href="#">Xapian::RSet</a> (A relevance set (R-Set) ) . . . . .	196

<a href="#">Xapian::RuntimeError</a> (The base class for exceptions indicating errors only detectable at runtime ) . . . . .	198
<a href="#">Xapian::SerialisationError</a> (Indicates an error in the std::string serialisation of an object ) . . . . .	199
<a href="#">Xapian::SimpleStopper</a> (Simple implementation of <a href="#">Stopper</a> class - this will suit most users ) . . . . .	201
<a href="#">Xapian::Sorter</a> (Virtual base class for sorter functor ) . . . . .	202
<a href="#">Xapian::Stem</a> (Class representing a stemming algorithm ) . . . . .	203
<a href="#">Xapian::StemImplementation</a> (Class representing a stemming algorithm implementation ) . . . . .	206
<a href="#">Xapian::Stopper</a> (Base class for stop-word decision functor ) . . . . .	207
<a href="#">Xapian::StringValueRangeProcessor</a> (Handle a string range ) . . . . .	208
<a href="#">Xapian::TermGenerator</a> (Parses a piece of text and generate terms ) . . . . .	210
<a href="#">Xapian::TermIterator</a> (An iterator pointing to items in a list of terms ) . . . . .	214
<a href="#">Xapian::TradWeight</a> ( <a href="#">Xapian::Weight</a> subclass implementing the traditional probabilistic formula ) . . . . .	217
<a href="#">Xapian::UnimplementedError</a> ( <a href="#">UnimplementedError</a> indicates an attempt to use an unimplemented feature ) . . . . .	221
<a href="#">Xapian::Utf8Iterator</a> (An iterator which returns <a href="#">Unicode</a> character values from a UTF-8 encoded string ) . . . . .	223
<a href="#">Xapian::ValueCountMatchSpy</a> (Class for counting the frequencies of values in the matching documents ) . . . . .	228
<a href="#">Xapian::ValueIterator</a> (Class for iterating over document values ) . . . . .	233
<a href="#">Xapian::ValueMapPostingSource</a> (A posting source which looks up weights in a map using values as the key ) . . . . .	236
<a href="#">Xapian::ValuePostingSource</a> (A posting source which generates weights from a value slot ) . . . . .	241
<a href="#">Xapian::ValueRangeProcessor</a> (Base class for value range processors ) . . . . .	247
<a href="#">Xapian::ValueSetMatchDecider</a> ( <a href="#">MatchDecider</a> filtering results based on whether document values are in a user-defined set ) . . . . .	248
<a href="#">Xapian::ValueWeightPostingSource</a> (A posting source which reads weights from a value slot ) . . . . .	250
<a href="#">Xapian::Weight</a> (Abstract base class for weighting schemes ) . . . . .	255
<a href="#">Xapian::WritableDatabase</a> (This class provides read/write access to a database ) . . . . .	261



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all documented files with brief descriptions:

xapian/error.h (Hierarchy of classes which <a href="#">Xapian</a> can throw as exceptions )	273
xapian/version.h (Define preprocessor symbols for the library version )	276
<a href="#">xapian.h</a> (Public interfaces for the <a href="#">Xapian</a> library )	278
xapian/compactor.h (Compact a database, or merge and compact several )	279
xapian/database.h (API for working with <a href="#">Xapian</a> databases )	280
xapian/dbfactory.h (Factory functions for constructing Database and WritableDatabase objects )	281
xapian/document.h (API for working with documents )	283
xapian/enquire.h (API for running queries )	284
xapian/errorhandler.h (Decide if a <a href="#">Xapian::Error</a> exception should be ignored )	286
xapian/expanddecider.h (Allow rejection of terms during ESet generation )	287
xapian/keymaker.h (Build key strings for MSet ordering or collapsing )	288
xapian/matchspy.h (MatchSpy implementation )	289
xapian/positioniterator.h (Classes for iterating through position lists )	290
xapian/postingiterator.h (Classes for iterating through posting lists )	291
xapian/postingsource.h (External sources of posting information )	292
xapian/query.h (Classes for representing a query )	293
xapian/queryparser.h (Parsing a user query string to build a <a href="#">Xapian::Query</a> object )	294
xapian/registry.h (Class for looking up user subclasses during unserialisation )	296
xapian/stem.h (Stemming algorithms )	297
xapian/termgenerator.h (Parse free text and generate terms )	298
xapian/termiterator.h (Classes for iterating through term lists )	299
xapian/types.h (Typedefs for <a href="#">Xapian</a> )	300
xapian/unicode.h (Unicode and UTF-8 related classes and functions )	302
xapian/valueiterator.h (Class for iterating over document values )	304
xapian/valuesetmatchdecider.h (MatchDecider subclass for filtering results by value )	305
xapian/weight.h (Weighting scheme API )	306



## Chapter 6

# Namespace Documentation

### 6.1 Xapian Namespace Reference

The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.

#### Namespaces

- namespace [Auto](#)  
*[Database](#) factory functions which determine the database type automatically.*
- namespace [Brass](#)  
*[Database](#) factory functions for the brass backend.*
- namespace [Chert](#)  
*[Database](#) factory functions for the chert backend.*
- namespace [Flint](#)  
*[Database](#) factory functions for the flint backend.*
- namespace [InMemory](#)  
*[Database](#) factory functions for the inmemory backend.*
- namespace [Remote](#)  
*[Database](#) factory functions for the remote backend.*
- namespace [Unicode](#)  
*Functions associated with handling [Unicode](#) characters.*

## Classes

- class [Error](#)  
*All exceptions thrown by [Xapian](#) are subclasses of [Xapian::Error](#).*
- class [LogicError](#)  
*The base class for exceptions indicating errors in the program logic.*
- class [RuntimeError](#)  
*The base class for exceptions indicating errors only detectable at runtime.*
- class [AssertionError](#)  
*[AssertionError](#) is thrown if a logical assertion inside [Xapian](#) fails.*
- class [InvalidArgumentError](#)  
*[InvalidArgumentError](#) indicates an invalid parameter value was passed to the API.*
- class [InvalidOperationError](#)  
*[InvalidOperationError](#) indicates the API was used in an invalid way.*
- class [UnimplementedError](#)  
*[UnimplementedError](#) indicates an attempt to use an unimplemented feature.*
- class [DatabaseError](#)  
*[DatabaseError](#) indicates some sort of database related error.*
- class [DatabaseCorruptError](#)  
*[DatabaseCorruptError](#) indicates database corruption was detected.*
- class [DatabaseCreateError](#)  
*[DatabaseCreateError](#) indicates a failure to create a database.*
- class [DatabaseLockError](#)  
*[DatabaseLockError](#) indicates failure to lock a database.*
- class [DatabaseModifiedError](#)  
*[DatabaseModifiedError](#) indicates a database was modified.*
- class [DatabaseOpeningError](#)  
*[DatabaseOpeningError](#) indicates failure to open a database.*
- class [DatabaseVersionError](#)  
*[DatabaseVersionError](#) indicates that a database is in an unsupported format.*
- class [DocNotFoundError](#)  
*Indicates an attempt to access a document not present in the database.*



- class [FeatureUnavailableError](#)  
*Indicates an attempt to use a feature which is unavailable.*
- class [InternalError](#)  
*[InternalError](#) indicates a runtime problem of some sort.*
- class [NetworkError](#)  
*Indicates a problem communicating with a remote database.*
- class [NetworkTimeoutError](#)  
*Indicates a timeout expired while communicating with a remote database.*
- class [QueryParserError](#)  
*Indicates a query string can't be parsed.*
- class [SerialisationError](#)  
*Indicates an error in the `std::string` serialisation of an object.*
- class [RangeError](#)  
*[RangeError](#) indicates an attempt to access outside the bounds of a container.*
- class [Compactor](#)  
*Compact a database, or merge and compact several.*
- class [Database](#)  
*This class is used to access a database, or a group of databases.*
- class [WritableDatabase](#)  
*This class provides read/write access to a database.*
- class [Document](#)  
*A handle representing a document in a [Xapian](#) database.*
- class [MSet](#)  
*A match set ([MSet](#)).*
- class [MSetIterator](#)  
*An iterator pointing to items in an [MSet](#).*
- class [ESet](#)  
*Class representing an ordered set of expand terms (an [ESet](#)).*
- class [ESetIterator](#)  
*Iterate through terms in the [ESet](#).*

- class [RSet](#)  
*A relevance set (R-Set).*
- class [MatchDecider](#)  
*Base class for matcher decision functor.*
- class [Enquire](#)  
*This class provides an interface to the information retrieval system for the purpose of searching.*
- class [ErrorHandler](#)  
*Decide if a [Xapian::Error](#) exception should be ignored.*
- class [ExpandDecider](#)  
*Virtual base class for expand decider functor.*
- class [ExpandDeciderAnd](#)  
*[ExpandDecider](#) subclass which rejects terms using two [ExpandDeciders](#).*
- class [ExpandDeciderFilterTerms](#)  
*[ExpandDecider](#) subclass which rejects terms in a specified list.*
- class [KeyMaker](#)  
*Virtual base class for key making functors.*
- class [MultiValueKeyMaker](#)  
*[KeyMaker](#) subclass which combines several values.*
- class [Sorter](#)  
*Virtual base class for sorter functor.*
- class [MultiValueSorter](#)  
*[Sorter](#) subclass which sorts by a several values.*
- class [MatchSpy](#)  
*Abstract base class for match spies.*
- class [ValueCountMatchSpy](#)  
*Class for counting the frequencies of values in the matching documents.*
- class [PositionIterator](#)  
*An iterator pointing to items in a list of positions.*
- class [PostingIterator](#)  
*An iterator pointing to items in a list of postings.*

- class [PostingSource](#)  
*Base class which provides an "external" source of postings.*
- class [ValuePostingSource](#)  
*A posting source which generates weights from a value slot.*
- class [ValueWeightPostingSource](#)  
*A posting source which reads weights from a value slot.*
- class [DecreasingValueWeightPostingSource](#)  
*Read weights from a value which is known to decrease as docid increases.*
- class [ValueMapPostingSource](#)  
*A posting source which looks up weights in a map using values as the key.*
- class [FixedWeightPostingSource](#)  
*A posting source which returns a fixed weight for all documents.*
- class [Query](#)  
*Class representing a query.*
- class [Stopper](#)  
*Base class for stop-word decision functor.*
- class [SimpleStopper](#)  
*Simple implementation of [Stopper](#) class - this will suit most users.*
- struct [ValueRangeProcessor](#)  
*Base class for value range processors.*
- class [StringValueRangeProcessor](#)  
*Handle a string range.*
- class [DateValueRangeProcessor](#)  
*Handle a date range.*
- class [NumberValueRangeProcessor](#)  
*Handle a number range.*
- class [QueryParser](#)  
*Build a [Xapian::Query](#) object from a user query string.*
- class [Registry](#)  
*[Registry](#) for user subclasses.*
- struct [StemImplementation](#)

*Class representing a stemming algorithm implementation.*

- class [Stem](#)  
*Class representing a stemming algorithm.*
- class [TermGenerator](#)  
*Parses a piece of text and generate terms.*
- class [TermIterator](#)  
*An iterator pointing to items in a list of terms.*
- class [Utf8Iterator](#)  
*An iterator which returns [Unicode](#) character values from a UTF-8 encoded string.*
- class [ValueIterator](#)  
*Class for iterating over document values.*
- class [ValueSetMatchDecider](#)  
*[MatchDecider](#) filtering results based on whether document values are in a user-defined set.*
- class [Weight](#)  
*Abstract base class for weighting schemes.*
- class [BoolWeight](#)  
*Class implementing a "boolean" weighting scheme.*
- class [BM25Weight](#)  
*[Xapian::Weight](#) subclass implementing the BM25 probabilistic formula.*
- class [TradWeight](#)  
*[Xapian::Weight](#) subclass implementing the traditional probabilistic formula.*

## Typedefs

- typedef unsigned [doccount](#)  
*A count of documents.*
- typedef int [doccount\\_diff](#)  
*A signed difference between two counts of documents.*
- typedef unsigned [docid](#)  
*A unique identifier for a document.*
- typedef double [doclength](#)

*A normalised document length.*

- typedef int [percent](#)  
*The percentage score for a document in an [MSet](#).*
- typedef unsigned [termcount](#)  
*A counts of terms.*
- typedef int [termcount\\_diff](#)  
*A signed difference between two counts of terms.*
- typedef unsigned [termpos](#)  
*A term position within a document or query.*
- typedef int [termpos\\_diff](#)  
*A signed difference between two term positions.*
- typedef unsigned [timeout](#)  
*A timeout value in milliseconds.*
- typedef unsigned [valueno](#)  
*The number for a value slot in a document.*
- typedef int [valueno\\_diff](#)  
*A signed difference between two value slot numbers.*
- typedef double [weight](#)  
*The weight of a document or term.*

## Functions

- bool [operator==](#) (const [MSetIterator](#) &a, const [MSetIterator](#) &b)  
*Equality test for [MSetIterator](#) objects.*
- bool [operator!=](#) (const [MSetIterator](#) &a, const [MSetIterator](#) &b)  
*Inequality test for [MSetIterator](#) objects.*
- bool [operator==](#) (const [ESetIterator](#) &a, const [ESetIterator](#) &b)  
*Equality test for [ESetIterator](#) objects.*
- bool [operator!=](#) (const [ESetIterator](#) &a, const [ESetIterator](#) &b)  
*Inequality test for [ESetIterator](#) objects.*
- bool [operator==](#) (const [PositionIterator](#) &a, const [PositionIterator](#) &b)  
*Test equality of two [PositionIterators](#).*

- bool `operator!=` (const [PositionIterator](#) &a, const [PositionIterator](#) &b)  
*Test inequality of two PositionIterators.*
- bool `operator==` (const [PostingIterator](#) &a, const [PostingIterator](#) &b)  
*Test equality of two PostingIterators.*
- bool `operator!=` (const [PostingIterator](#) &a, const [PostingIterator](#) &b)  
*Test inequality of two PostingIterators.*
- std::string `sortable_serialise` (double value)  
*Convert a floating point number to a string, preserving sort order.*
- double `sortable_unserialise` (const std::string &value)  
*Convert a string encoded using sortable\_serialise back to a floating point number.*
- bool `operator==` (const [TermIterator](#) &a, const [TermIterator](#) &b)  
*Equality test for TermIterator objects.*
- bool `operator!=` (const [TermIterator](#) &a, const [TermIterator](#) &b)  
*Inequality test for TermIterator objects.*
- bool `operator==` (const [ValueIterator](#) &a, const [ValueIterator](#) &b)  
*Equality test for ValueIterator objects.*
- bool `operator!=` (const [ValueIterator](#) &a, const [ValueIterator](#) &b)  
*Inequality test for ValueIterator objects.*
- const char \* `version_string` ()  
*Report the version string of the library which the program is linked with.*
- int `major_version` ()  
*Report the major version of the library which the program is linked with.*
- int `minor_version` ()  
*Report the minor version of the library which the program is linked with.*
- int `revision` ()  
*Report the revision of the library which the program is linked with.*

## Variables

- const int `DB_CREATE_OR_OPEN` = 1  
*Open for read/write; create if no db exists.*

- const int `DB_CREATE` = 2  
*Create a new database; fail if db exists.*
- const int `DB_CREATE_OR_OVERWRITE` = 3  
*Overwrite existing db; create if none exists.*
- const int `DB_OPEN` = 4  
*Open for read/write; fail if no db exists.*
- const `valueno BAD_VALUENO` = static\_cast<valueno>(-1)  
*Reserved value to indicate "no valueno".*

### 6.1.1 Detailed Description

The `Xapian` namespace contains public interfaces for the `Xapian` library.

### 6.1.2 Typedef Documentation

#### 6.1.2.1 typedef unsigned Xapian::doccount

A count of documents.

This is used to hold values such as the number of documents in a database and the frequency of a term in the database.

#### 6.1.2.2 typedef int Xapian::doccount\_diff

A signed difference between two counts of documents.

This is used by the `Xapian` classes which are STL containers of documents for "difference\_type".

#### 6.1.2.3 typedef unsigned Xapian::docid

A unique identifier for a document.

Docid 0 is invalid, providing an "out of range" value which can be used to mean "not a valid document".

#### 6.1.2.4 typedef double Xapian::doclength

A normalised document length.

The normalised document length is the document length divided by the average document length in the database.

**6.1.2.5 typedef int Xapian::percent**

The percentage score for a document in an [MSet](#).

**6.1.2.6 typedef unsigned Xapian::termcount**

A counts of terms.

This is used to hold values such as the Within [Document](#) Frequency (wdf).

**6.1.2.7 typedef int Xapian::termcount\_diff**

A signed difference between two counts of terms.

This is used by the [Xapian](#) classes which are STL containers of terms for "difference\_type".

**6.1.2.8 typedef int Xapian::termpos\_diff**

A signed difference between two term positions.

This is used by the [Xapian](#) classes which are STL containers of positions for "difference\_type".

**6.1.2.9 typedef unsigned Xapian::timeout**

A timeout value in milliseconds.

There are 1000 milliseconds in a second, so for example, to set a timeout of 5 seconds use 5000.

**6.1.2.10 typedef unsigned Xapian::valueno**

The number for a value slot in a document.

Any value slot number except [Xapian::BAD\\_VALUENO](#) is valid.

**6.1.2.11 typedef int Xapian::valueno\_diff**

A signed difference between two value slot numbers.

This is used by the [Xapian](#) classes which are STL containers of values for "difference\_type".

**6.1.2.12 typedef double Xapian::weight**

The weight of a document or term.



### 6.1.3 Function Documentation

#### 6.1.3.1 `int Xapian::major_version ()`

Report the major version of the library which the program is linked with.

This may be different to the version compiled against (given by `XAPIAN_MAJOR_VERSION`) if shared libraries are being used.

#### 6.1.3.2 `int Xapian::minor_version ()`

Report the minor version of the library which the program is linked with.

This may be different to the version compiled against (given by `XAPIAN_MINOR_VERSION`) if shared libraries are being used.

#### 6.1.3.3 `int Xapian::revision ()`

Report the revision of the library which the program is linked with.

This may be different to the version compiled against (given by `XAPIAN_REVISION`) if shared libraries are being used.

#### 6.1.3.4 `std::string Xapian::sortable_serialise (double value)`

Convert a floating point number to a string, preserving sort order.

This method converts a floating point number to a string, suitable for using as a value for numeric range restriction, or for use as a sort key.

The conversion is platform independent.

The conversion attempts to ensure that, for any pair of values supplied to the conversion algorithm, the result of comparing the original values (with a numeric comparison operator) will be the same as the result of comparing the resulting values (with a string comparison operator). On platforms which represent doubles with the precisions specified by IEEE\_754, this will be the case: if the representation of doubles is more precise, it is possible that two very close doubles will be mapped to the same string, so will compare equal.

Note also that both zero and -zero will be converted to the same representation: since these compare equal, this satisfies the comparison constraint, but it's worth knowing this if you wish to use the encoding in some situation where this distinction matters.

Handling of NaN isn't (currently) guaranteed to be sensible.

#### 6.1.3.5 `double Xapian::sortable_unserialise (const std::string & value)`

Convert a string encoded using *sortable\_serialise* back to a floating point number.

This expects the input to be a string produced by *sortable\_serialise()*. If the input is not such a string, the value returned is undefined (but no error will be thrown).

The result of the conversion will be exactly the value which was supplied to [sortable\\_serialise\(\)](#) when making the string on platforms which represent doubles with the precisions specified by IEEE\_754, but may be a different (nearby) value on other platforms.

#### **6.1.3.6 `const char* Xapian::version_string ()`**

Report the version string of the library which the program is linked with.

This may be different to the version compiled against (given by XAPIAN\_VERSION) if shared libraries are being used.

### **6.1.4 Variable Documentation**

#### **6.1.4.1 `const valueno Xapian::BAD_VALUENO = static_cast<valueno>(-1)`**

Reserved value to indicate "no valueno".

#### **6.1.4.2 `const int Xapian::DB_CREATE = 2`**

Create a new database; fail if db exists.

#### **6.1.4.3 `const int Xapian::DB_CREATE_OR_OPEN = 1`**

Open for read/write; create if no db exists.

#### **6.1.4.4 `const int Xapian::DB_CREATE_OR_OVERWRITE = 3`**

Overwrite existing db; create if none exists.

#### **6.1.4.5 `const int Xapian::DB_OPEN = 4`**

Open for read/write; fail if no db exists.

## 6.2 Xapian::Auto Namespace Reference

[Database](#) factory functions which determine the database type automatically.

### Functions

- [Database open\\_stub](#) (const std::string &file)  
*Construct a [Database](#) object for a stub database file.*
- [WritableDatabase open\\_stub](#) (const std::string &file, int action)  
*Construct a [WritableDatabase](#) object for a stub database file.*

### 6.2.1 Detailed Description

[Database](#) factory functions which determine the database type automatically.

### 6.2.2 Function Documentation

#### 6.2.2.1 WritableDatabase Xapian::Auto::open\_stub (const std::string &file, int action)

Construct a [WritableDatabase](#) object for a stub database file.

The stub database file must contain serialised parameters for exactly one database.

##### Parameters:

*file* pathname of the stub database file.

#### 6.2.2.2 Database Xapian::Auto::open\_stub (const std::string &file)

Construct a [Database](#) object for a stub database file.

The stub database file contains serialised parameters for one or more databases.

##### Parameters:

*file* pathname of the stub database file.

## 6.3 Xapian::Brass Namespace Reference

[Database](#) factory functions for the brass backend.

### Functions

- [Database open](#) (const std::string &dir)  
*Construct a [Database](#) object for read-only access to a [Brass](#) database.*
- [WritableDatabase open](#) (const std::string &dir, int action, int block\_size=8192)  
*Construct a [Database](#) object for update access to a [Brass](#) database.*

### 6.3.1 Detailed Description

[Database](#) factory functions for the brass backend.

### 6.3.2 Function Documentation

#### 6.3.2.1 WritableDatabase Xapian::Brass::open (const std::string & dir, int action, int block\_size = 8192)

Construct a [Database](#) object for update access to a [Brass](#) database.

##### Parameters:

*dir* pathname of the directory containing the database.

*action* determines handling of existing/non-existing database:

- [Xapian::DB\\_CREATE](#) fail if database already exist, otherwise create new database.
- [Xapian::DB\\_CREATE\\_OR\\_OPEN](#) open existing database, or create new database if none exists.
- [Xapian::DB\\_CREATE\\_OR\\_OVERWRITE](#) overwrite existing database, or create new database if none exists.
- [Xapian::DB\\_OPEN](#) open existing database, failing if none exists.

*block\_size* the Btree blocksize to use (in bytes), which must be a power of two between 2048 and 65536 (inclusive). The default (also used if an invalid value is passed) is 8192 bytes. This parameter is ignored when opening an existing database.

#### 6.3.2.2 Database Xapian::Brass::open (const std::string & dir)

Construct a [Database](#) object for read-only access to a [Brass](#) database.

#### Parameters:

*dir* pathname of the directory containing the database.

## 6.4 Xapian::Chert Namespace Reference

[Database](#) factory functions for the chert backend.

### Functions

- [Database open](#) (const std::string &dir)  
*Construct a [Database](#) object for read-only access to a [Chert](#) database.*
- [WritableDatabase open](#) (const std::string &dir, int action, int block\_size=8192)  
*Construct a [Database](#) object for update access to a [Chert](#) database.*

### 6.4.1 Detailed Description

[Database](#) factory functions for the chert backend.

### 6.4.2 Function Documentation

#### 6.4.2.1 WritableDatabase Xapian::Chert::open (const std::string & dir, int action, int block\_size = 8192)

Construct a [Database](#) object for update access to a [Chert](#) database.

##### Parameters:

*dir* pathname of the directory containing the database.

*action* determines handling of existing/non-existing database:

- [Xapian::DB\\_CREATE](#) fail if database already exist, otherwise create new database.
- [Xapian::DB\\_CREATE\\_OR\\_OPEN](#) open existing database, or create new database if none exists.
- [Xapian::DB\\_CREATE\\_OR\\_OVERWRITE](#) overwrite existing database, or create new database if none exists.
- [Xapian::DB\\_OPEN](#) open existing database, failing if none exists.

*block\_size* the Btree blocksize to use (in bytes), which must be a power of two between 2048 and 65536 (inclusive). The default (also used if an invalid value is passed) is 8192 bytes. This parameter is ignored when opening an existing database.

#### 6.4.2.2 Database Xapian::Chert::open (const std::string & dir)

Construct a [Database](#) object for read-only access to a [Chert](#) database.

### Parameters:

*dir* pathname of the directory containing the database.

## 6.5 Xapian::Flint Namespace Reference

[Database](#) factory functions for the flint backend.

### Functions

- [Database open](#) (const std::string &dir)  
*Construct a [Database](#) object for read-only access to a [Flint](#) database.*
- [WritableDatabase open](#) (const std::string &dir, int action, int block\_size=8192)  
*Construct a [Database](#) object for update access to a [Flint](#) database.*

### 6.5.1 Detailed Description

[Database](#) factory functions for the flint backend.

### 6.5.2 Function Documentation

#### 6.5.2.1 WritableDatabase Xapian::Flint::open (const std::string & dir, int action, int block\_size = 8192)

Construct a [Database](#) object for update access to a [Flint](#) database.

##### Parameters:

*dir* pathname of the directory containing the database.

*action* determines handling of existing/non-existing database:

- [Xapian::DB\\_CREATE](#) fail if database already exist, otherwise create new database.
- [Xapian::DB\\_CREATE\\_OR\\_OPEN](#) open existing database, or create new database if none exists.
- [Xapian::DB\\_CREATE\\_OR\\_OVERWRITE](#) overwrite existing database, or create new database if none exists.
- [Xapian::DB\\_OPEN](#) open existing database, failing if none exists.

*block\_size* the Btree blocksize to use (in bytes), which must be a power of two between 2048 and 65536 (inclusive). The default (also used if an invalid value is passed) is 8192 bytes. This parameter is ignored when opening an existing database.

#### 6.5.2.2 Database Xapian::Flint::open (const std::string & dir)

Construct a [Database](#) object for read-only access to a [Flint](#) database.



**Parameters:**

*dir* pathname of the directory containing the database.

## 6.6 Xapian::InMemory Namespace Reference

[Database](#) factory functions for the inmemory backend.

### Functions

- [WritableDatabase](#) `open ()`

*Construct a [WritableDatabase](#) object for a new, empty [InMemory](#) database.*

### 6.6.1 Detailed Description

[Database](#) factory functions for the inmemory backend.

### 6.6.2 Function Documentation

#### 6.6.2.1 WritableDatabase Xapian::InMemory::open ()

Construct a [WritableDatabase](#) object for a new, empty [InMemory](#) database.

Only a writable [InMemory](#) database can be created, since a read-only one would always remain empty.

## 6.7 Xapian::Remote Namespace Reference

[Database](#) factory functions for the remote backend.

### Functions

- [Database open](#) (const std::string &host, unsigned int port, [Xapian::timeout timeout=10000](#), [Xapian::timeout connect\\_timeout=10000](#))  
*Construct a [Database](#) object for read-only access to a remote database accessed via a TCP connection.*
- [WritableDatabase open\\_writable](#) (const std::string &host, unsigned int port, [Xapian::timeout timeout=0](#), [Xapian::timeout connect\\_timeout=10000](#))  
*Construct a [WritableDatabase](#) object for update access to a remote database accessed via a TCP connection.*
- [Database open](#) (const std::string &program, const std::string &args, [Xapian::timeout timeout=10000](#))  
*Construct a [Database](#) object for read-only access to a remote database accessed via a program.*
- [WritableDatabase open\\_writable](#) (const std::string &program, const std::string &args, [Xapian::timeout timeout=0](#))  
*Construct a [WritableDatabase](#) object for update access to a remote database accessed via a program.*

### 6.7.1 Detailed Description

[Database](#) factory functions for the remote backend.

### 6.7.2 Function Documentation

#### 6.7.2.1 Database Xapian::Remote::open (const std::string & *program*, const std::string & *args*, Xapian::timeout *timeout* = 10000)

Construct a [Database](#) object for read-only access to a remote database accessed via a program.

Access to the remote database is done by running an external program and communicating with it on stdin/stdout.

#### Parameters:

*program* the external program to run.

*args* space-separated list of arguments to pass to program.

***timeout*** timeout in milliseconds. If this timeout is exceeded for any individual operation on the remote database then [Xapian::NetworkTimeoutError](#) is thrown. A timeout of 0 means don't timeout. (Default is 10000ms, which is 10 seconds).

#### 6.7.2.2 Database `Xapian::Remote::open` (`const std::string & host`, `unsigned int port`, `Xapian::timeout timeout = 10000`, `Xapian::timeout connect_timeout = 10000`)

Construct a [Database](#) object for read-only access to a remote database accessed via a TCP connection.

Access to the remote database is via a TCP connection to the specified host and port.

##### Parameters:

***host*** hostname to connect to.

***port*** port number to connect to.

***timeout*** timeout in milliseconds. If this timeout is exceeded for any individual operation on the remote database then [Xapian::NetworkTimeoutError](#) is thrown. A timeout of 0 means don't timeout. (Default is 10000ms, which is 10 seconds).

***connect\_timeout*** timeout to use when connecting to the server. If this timeout is exceeded then [Xapian::NetworkTimeoutError](#) is thrown. A timeout of 0 means don't timeout. (Default is 10000ms, which is 10 seconds).

#### 6.7.2.3 WritableDatabase `Xapian::Remote::open_writable` (`const std::string & program`, `const std::string & args`, `Xapian::timeout timeout = 0`)

Construct a [WritableDatabase](#) object for update access to a remote database accessed via a program.

Access to the remote database is done by running an external program and communicating with it on stdin/stdout.

##### Parameters:

***program*** the external program to run.

***args*** space-separated list of arguments to pass to program.

***timeout*** timeout in milliseconds. If this timeout is exceeded for any individual operation on the remote database then [Xapian::NetworkTimeoutError](#) is thrown. (Default is 0, which means don't timeout).

#### 6.7.2.4 WritableDatabase Xapian::Remote::open\_writable (const std::string & *host*, unsigned int *port*, Xapian::timeout *timeout* = 0, Xapian::timeout *connect\_timeout* = 10000)

Construct a [WritableDatabase](#) object for update access to a remote database accessed via a TCP connection.

Access to the remote database is via a TCP connection to the specified host and port.

##### Parameters:

*host* hostname to connect to.

*port* port number to connect to.

*timeout* timeout in milliseconds. If this timeout is exceeded for any individual operation on the remote database then [Xapian::NetworkTimeoutError](#) is thrown. (Default is 0, which means don't timeout).

*connect\_timeout* timeout to use when connecting to the server. If this timeout is exceeded then [Xapian::NetworkTimeoutError](#) is thrown. A timeout of 0 means don't timeout. (Default is 10000ms, which is 10 seconds).

## 6.8 Xapian::Unicode Namespace Reference

Functions associated with handling [Unicode](#) characters.

### Enumerations

- enum [category](#)

*Each Unicode character is in exactly one of these categories.*

### Functions

- unsigned [nonascii\\_to\\_utf8](#) (unsigned ch, char \*buf)  
*Convert a single non-ASCII [Unicode](#) character to UTF-8.*

- unsigned [to\\_utf8](#) (unsigned ch, char \*buf)  
*Convert a single [Unicode](#) character to UTF-8.*

- void [append\\_utf8](#) (std::string &s, unsigned ch)  
*Append the UTF-8 representation of a single [Unicode](#) character to a std::string.*

- [category](#) [get\\_category](#) (unsigned ch)  
*Return the category which a given [Unicode](#) character falls into.*

- bool [is\\_wordchar](#) (unsigned ch)  
*Test if a given [Unicode](#) character is "word character".*

- bool [is\\_whitespace](#) (unsigned ch)  
*Test if a given [Unicode](#) character is a whitespace character.*

- bool [is\\_currency](#) (unsigned ch)  
*Test if a given [Unicode](#) character is a currency symbol.*

- unsigned [tolower](#) (unsigned ch)  
*Convert a [Unicode](#) character to lowercase.*

- unsigned [toupper](#) (unsigned ch)  
*Convert a [Unicode](#) character to uppercase.*

- std::string [tolower](#) (const std::string &term)  
*Convert a UTF-8 std::string to lowercase.*

- std::string [toupper](#) (const std::string &term)  
*Convert a UTF-8 std::string to uppercase.*

## 6.8.1 Detailed Description

Functions associated with handling [Unicode](#) characters.

## 6.8.2 Enumeration Type Documentation

### 6.8.2.1 enum Xapian::Unicode::category

Each [Unicode](#) character is in exactly one of these categories.

## 6.8.3 Function Documentation

### 6.8.3.1 unsigned Xapian::Unicode::nonascii\_to\_utf8 (unsigned *ch*, char \* *buf*)

Convert a single non-ASCII [Unicode](#) character to UTF-8.

This is intended mainly as a helper method for [to\\_utf8\(\)](#).

The character *ch* (which must be > 128) is written to the buffer *buf* and the length of the resultant UTF-8 character is returned.

NB *buf* must have space for (at least) 4 bytes.

Referenced by [to\\_utf8\(\)](#).

### 6.8.3.2 unsigned Xapian::Unicode::to\_utf8 (unsigned *ch*, char \* *buf*) [inline]

Convert a single [Unicode](#) character to UTF-8.

The character *ch* is written to the buffer *buf* and the length of the resultant UTF-8 character is returned.

NB *buf* must have space for (at least) 4 bytes.

References [nonascii\\_to\\_utf8\(\)](#).

Referenced by [append\\_utf8\(\)](#).





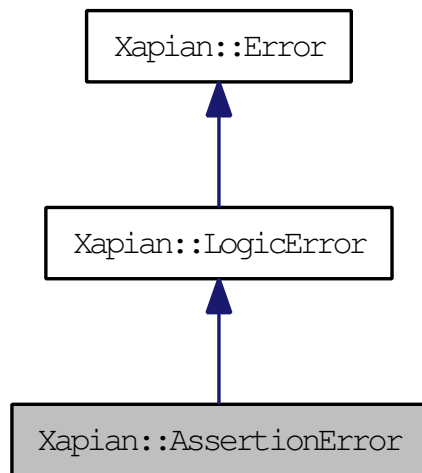
## Chapter 7

# Class Documentation

### 7.1 Xapian::AssertionError Class Reference

[AssertionError](#) is thrown if a logical assertion inside [Xapian](#) fails.

Inheritance diagram for Xapian::AssertionError:



#### Public Member Functions

- [AssertionError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_=0)  
*General purpose constructor which allows setting errno.*
- [AssertionError](#) (const std::string &msg\_, int errno\_)  
*Construct from message and errno value.*

### 7.1.1 Detailed Description

[AssertionError](#) is thrown if a logical assertion inside [Xapian](#) fails.

In a debug build of [Xapian](#), a failed assertion in the core library code will cause [AssertionError](#) to be thrown.

This represents a bug in [Xapian](#) (either an invariant, precondition, etc has been violated, or the assertion is incorrect!)

### 7.1.2 Constructor & Destructor Documentation

**7.1.2.1 [Xapian::AssertionError::AssertionError](#) (const std::string & *msg\_*, const std::string & *context\_* = std::string(), int *errno\_* = 0) [inline, explicit]**

General purpose constructor which allows setting errno.

**7.1.2.2 [Xapian::AssertionError::AssertionError](#) (const std::string & *msg\_*, int *errno\_*) [inline]**

Construct from message and errno value.

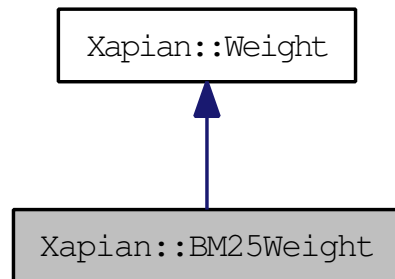
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.2 Xapian::BM25Weight Class Reference

[Xapian::Weight](#) subclass implementing the BM25 probabilistic formula.

Inheritance diagram for Xapian::BM25Weight:



### Public Member Functions

- [BM25Weight](#) (double k1, double k2, double k3, double b, double min\_normlen)  
*Construct a [BM25Weight](#).*
- std::string [name](#) () const  
*Return the name of this weighting scheme.*
- std::string [serialise](#) () const  
*Return this object's parameters serialised as a single string.*
- [BM25Weight](#) \* [unserialise](#) (const std::string &s) const  
*Unserialise parameters.*
- [Xapian::weight](#) [get\\_sumpart](#) ([Xapian::termcount](#) wdf, [Xapian::termcount](#) doclen) const  
*Calculate the weight contribution for this object's term to a document.*
- [Xapian::weight](#) [get\\_maxpart](#) () const  
*Return an upper bound on what [get\\_sumpart\(\)](#) can return for any document.*
- [Xapian::weight](#) [get\\_sumextra](#) ([Xapian::termcount](#) doclen) const  
*Calculate the term-independent weight component for a document.*
- [Xapian::weight](#) [get\\_maxextra](#) () const  
*Return an upper bound on what [get\\_sumextra\(\)](#) can return for any document.*

### 7.2.1 Detailed Description

[Xapian::Weight](#) subclass implementing the BM25 probabilistic formula.

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 `Xapian::BM25Weight::BM25Weight (double k1, double k2, double k3, double b, double min_normlen) [inline]`

Construct a [BM25Weight](#).

##### Parameters:

- k1* A non-negative parameter controlling how influential within-document-frequency (wdf) is. *k1*=0 means that wdf doesn't affect the weights. The larger *k1* is, the more wdf influences the weights. (default 1)
- k2* A non-negative parameter which controls the strength of a correction factor which depends upon query length and normalised document length. *k2*=0 disable this factor; larger *k2* makes it stronger. (default 0)
- k3* A non-negative parameter controlling how influential within-query-frequency (wqf) is. *k3*=0 means that wqf doesn't affect the weights. The larger *k3* is, the more wqf influences the weights. (default 1)
- b* A parameter between 0 and 1, controlling how strong the document length normalisation of wdf is. 0 means no normalisation; 1 means full normalisation. (default 0.5)
- min\_normlen* A parameter specifying a minimum value for normalised document length. Normalised document length values less than this will be clamped to this value, helping to prevent very short documents getting large weights. (default 0.5)

### 7.2.3 Member Function Documentation

#### 7.2.3.1 `Xapian::weight Xapian::BM25Weight::get_maxextra () const` [virtual]

Return an upper bound on what [get\\_sumextra\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implements [Xapian::Weight](#).

#### 7.2.3.2 `Xapian::weight Xapian::BM25Weight::get_maxpart () const` [virtual]

Return an upper bound on what [get\\_sumpart\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implements [Xapian::Weight](#).

#### 7.2.3.3 Xapian::weight Xapian::BM25Weight::get\_sumextra (Xapian::termcount *doclen*) const [virtual]

Calculate the term-independent weight component for a document.

The parameter gives information about the document which may be used in the calculations:

**Parameters:**

*doclen* The document's length (unnormalised).

Implements [Xapian::Weight](#).

#### 7.2.3.4 Xapian::weight Xapian::BM25Weight::get\_sumpart (Xapian::termcount *wdf*, Xapian::termcount *doclen*) const [virtual]

Calculate the weight contribution for this object's term to a document.

The parameters give information about the document which may be used in the calculations:

**Parameters:**

*wdf* The within document frequency of the term in the document.

*doclen* The document's length (unnormalised).

Implements [Xapian::Weight](#).

#### 7.2.3.5 std::string Xapian::BM25Weight::name () const [virtual]

Return the name of this weighting scheme.

This name is used by the remote backend. It is passed along with the serialised parameters to the remote server so that it knows which class to create.

Return the full namespace-qualified name of your class here - if your class is called FooWeight, return "FooWeight" from this method ([Xapian::BM25Weight](#) returns "Xapian::BM25Weight" here).

If you don't want to support the remote backend, you can use the default implementation which simply returns an empty string.

Reimplemented from [Xapian::Weight](#).

### 7.2.3.6 `std::string Xapian::BM25Weight::serialise () const` [virtual]

Return this object's parameters serialised as a single string.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::Weight](#).

### 7.2.3.7 `BM25Weight* Xapian::BM25Weight::unserialise (const std::string & s) const` [virtual]

Unserialise parameters.

This method unserialises parameters serialised by the [serialise\(\)](#) method and allocates and returns a new object initialised with them.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented from [Xapian::Weight](#).

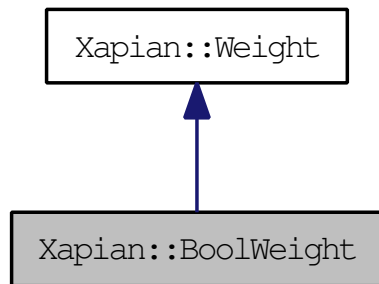
The documentation for this class was generated from the following file:

- [xapian/weight.h](#)

## 7.3 Xapian::BoolWeight Class Reference

Class implementing a "boolean" weighting scheme.

Inheritance diagram for Xapian::BoolWeight:



### Public Member Functions

- [BoolWeight \(\)](#)  
*Construct a [BoolWeight](#).*
- [std::string name \(\) const](#)  
*Return the name of this weighting scheme.*
- [std::string serialise \(\) const](#)  
*Return this object's parameters serialised as a single string.*
- [BoolWeight \\* unserialise \(const std::string &s\) const](#)  
*Unserialise parameters.*
- [Xapian::weight get\\_sumpart \(Xapian::termcount wdf, Xapian::termcount doclen\) const](#)  
*Calculate the weight contribution for this object's term to a document.*
- [Xapian::weight get\\_maxpart \(\) const](#)  
*Return an upper bound on what [get\\_sumpart\(\)](#) can return for any document.*
- [Xapian::weight get\\_sumextra \(Xapian::termcount doclen\) const](#)  
*Calculate the term-independent weight component for a document.*
- [Xapian::weight get\\_maxextra \(\) const](#)  
*Return an upper bound on what [get\\_sumextra\(\)](#) can return for any document.*

### 7.3.1 Detailed Description

Class implementing a "boolean" weighting scheme.

This weighting scheme gives all documents zero weight.

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 `Xapian::BoolWeight::BoolWeight ()` [inline]

Construct a [BoolWeight](#).

### 7.3.3 Member Function Documentation

#### 7.3.3.1 `Xapian::weight Xapian::BoolWeight::get_maxextra () const` [virtual]

Return an upper bound on what [get\\_sumextra\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implements [Xapian::Weight](#).

#### 7.3.3.2 `Xapian::weight Xapian::BoolWeight::get_maxpart () const` [virtual]

Return an upper bound on what [get\\_sumpart\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implements [Xapian::Weight](#).

#### 7.3.3.3 `Xapian::weight Xapian::BoolWeight::get_sumextra (Xapian::termcount doclen) const` [virtual]

Calculate the term-independent weight component for a document.

The parameter gives information about the document which may be used in the calculations:

**Parameters:**

*doclen* The document's length (unnormalised).

Implements [Xapian::Weight](#).



#### 7.3.3.4 Xapian::weight Xapian::BoolWeight::get\_sumpart (Xapian::termcount *wdf*, Xapian::termcount *doclen*) const [virtual]

Calculate the weight contribution for this object's term to a document.

The parameters give information about the document which may be used in the calculations:

##### Parameters:

*wdf* The within document frequency of the term in the document.

*doclen* The document's length (unnormalised).

Implements [Xapian::Weight](#).

#### 7.3.3.5 std::string Xapian::BoolWeight::name () const [virtual]

Return the name of this weighting scheme.

This name is used by the remote backend. It is passed along with the serialised parameters to the remote server so that it knows which class to create.

Return the full namespace-qualified name of your class here - if your class is called FooWeight, return "FooWeight" from this method ([Xapian::BM25Weight](#) returns "Xapian::BM25Weight" here).

If you don't want to support the remote backend, you can use the default implementation which simply returns an empty string.

Reimplemented from [Xapian::Weight](#).

#### 7.3.3.6 std::string Xapian::BoolWeight::serialise () const [virtual]

Return this object's parameters serialised as a single string.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::Weight](#).

#### 7.3.3.7 BoolWeight\* Xapian::BoolWeight::unserialise (const std::string & s) const [virtual]

Unserialise parameters.

This method unserialises parameters serialised by the [serialise\(\)](#) method and allocates and returns a new object initialised with them.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented from [Xapian::Weight](#).

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

- [xapian/weight.h](#)

## 7.4 Xapian::Compactor Class Reference

Compact a database, or merge and compact several.

### Public Member Functions

- void [set\\_block\\_size](#) (size\_t block\_size)  
*Set the block size to use for tables in the output database.*
- void [set\\_renumber](#) (bool renumber)  
*Set whether to preserve existing document id values.*
- void [set\\_multipass](#) (bool multipass)  
*Set whether to merge postlists in multiple passes.*
- void [set\\_compaction\\_level](#) (compaction\_level compaction)  
*Set the compaction level.*
- void [set\\_destdir](#) (const std::string &destdir)  
*Set where to write the output.*
- void [add\\_source](#) (const std::string &srcdir)  
*Add a source database.*
- void [compact](#) ()  
*Perform the actual compaction/merging operation.*
- virtual void [set\\_status](#) (const std::string &table, const std::string &status)  
*Update progress.*
- virtual std::string [resolve\\_duplicate\\_metadata](#) (const std::string &key, size\_t num\_tags, const std::string tags[ ])  
*Resolve multiple user metadata entries with the same key.*

### 7.4.1 Detailed Description

Compact a database, or merge and compact several.

### 7.4.2 Member Function Documentation

#### 7.4.2.1 void Xapian::Compactor::add\_source (const std::string &srcdir)

Add a source database.

#### 7.4.2.2 **virtual std::string Xpian::Compactor::resolve\_duplicate\_metadata** (const std::string & key, size\_t num\_tags, const std::string tags[ ]) [virtual]

Resolve multiple user metadata entries with the same key.

When merging, if the same user metadata key is set in more than one input, then this method is called to allow this to be resolving in an appropriate way.

The default implementation just returns tags[0].

For multipass this will currently get called multiple times for the same key if there are duplicates to resolve in each pass, but this may change in the future.

#### 7.4.2.3 **void Xpian::Compactor::set\_block\_size** (size\_t block\_size)

Set the block size to use for tables in the output database.

Valid block sizes are currently powers of two between 2048 and 65536, with the default being 8192, but the valid sizes and default may change in the future.

#### 7.4.2.4 **void Xpian::Compactor::set\_compaction\_level** (compaction\_level compaction)

Set the compaction level.

Values are:

- Xpian::Compactor::STANDARD - Don't split items unnecessarily.
- Xpian::Compactor::FULL - Split items whenever it saves space (the default).
- Xpian::Compactor::FULLER - Allow oversize items to save more space.

FULLER isn't recommended if you ever plan to update the compacted database.

#### 7.4.2.5 **void Xpian::Compactor::set\_destdir** (const std::string & destdir)

Set where to write the output.

This can be the same as an input if that input is a stub database (in which case it will be updated atomically).

#### 7.4.2.6 **void Xpian::Compactor::set\_multipass** (bool multipass)

Set whether to merge postlists in multiple passes.

Default is false. If set to true and merging more than 3 databases, merge the postlists in multiple passes, which is generally faster but requires more disk space for temporary files.

**7.4.2.7 void Xapian::Compactor::set\_renumber (bool *renumber*)**

Set whether to preserve existing document id values.

The default is true, which means that document ids will be renumbered - currently by applying the same offset to all the document ids in a particular source database.

If false, then the document ids must be unique over all source databases. Currently the ranges of document ids in each source must not overlap either, though this restriction may be removed in the future.

**7.4.2.8 virtual void Xapian::Compactor::set\_status (const std::string & *table*, const std::string & *status*)** [virtual]

Update progress.

Subclass this method if you want to get progress updates during compaction. This is called for each table first with empty status, And then one or more times with non-empty status.

The default implementation does nothing.

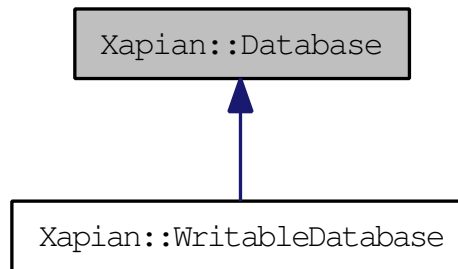
The documentation for this class was generated from the following file:

- [xapian/compactor.h](#)

## 7.5 Xapian::Database Class Reference

This class is used to access a database, or a group of databases.

Inheritance diagram for Xapian::Database:



### Public Member Functions

- void [add\\_database](#) (const [Database](#) &database)  
*Add an existing database (or group of databases) to those accessed by this object.*
- [Database](#) ()  
*Create a [Database](#) with no databases in.*
- [Database](#) (const std::string &path)  
*Open a [Database](#), automatically determining the database backend to use.*
- virtual [~Database](#) ()  
*Destroy this handle on the database.*
- [Database](#) (const [Database](#) &other)  
*Copying is allowed.*
- void [operator=](#) (const [Database](#) &other)  
*Assignment is allowed.*
- void [reopen](#) ()  
*Re-open the database.*
- virtual void [close](#) ()  
*Close the database.*
- virtual std::string [get\\_description](#) () const  
*Return a string describing this object.*
- [PostingIterator](#) [postlist\\_begin](#) (const std::string &tname) const

*An iterator pointing to the start of the postlist for a given term.*

- [PostingIterator postlist\\_end](#) (const std::string &) const  
*Corresponding end iterator to [postlist\\_begin\(\)](#).*
- [TermIterator termlist\\_begin](#) (Xapian::docid did) const  
*An iterator pointing to the start of the termlist for a given document.*
- [TermIterator termlist\\_end](#) (Xapian::docid) const  
*Corresponding end iterator to [termlist\\_begin\(\)](#).*
- bool [has\\_positions](#) () const  
*Does this database have any positional information?*
- [PositionIterator positionlist\\_begin](#) (Xapian::docid did, const std::string &name) const  
*An iterator pointing to the start of the position list for a given term in a given document.*
- [PositionIterator positionlist\\_end](#) (Xapian::docid, const std::string &) const  
*Corresponding end iterator to [positionlist\\_begin\(\)](#).*
- [TermIterator allterms\\_begin](#) () const  
*An iterator which runs across all terms in the database.*
- [TermIterator allterms\\_end](#) () const  
*Corresponding end iterator to [allterms\\_begin\(\)](#).*
- [TermIterator allterms\\_begin](#) (const std::string &prefix) const  
*An iterator which runs across all terms with a given prefix.*
- [TermIterator allterms\\_end](#) (const std::string &) const  
*Corresponding end iterator to [allterms\\_begin\(prefix\)](#).*
- [Xapian::doccount get\\_doccount](#) () const  
*Get the number of documents in the database.*
- [Xapian::docid get\\_lastdocid](#) () const  
*Get the highest document id which has been used in the database.*
- [Xapian::doclength get\\_avlength](#) () const  
*Get the average length of the documents in the database.*
- [Xapian::doccount get\\_termfreq](#) (const std::string &name) const  
*Get the number of documents in the database indexed by a given term.*

- `bool term_exists (const std::string &tname) const`  
*Check if a given term exists in the database.*
- `Xapian::termcount get_collection_freq (const std::string &tname) const`  
*Return the total number of occurrences of the given term.*
- `Xapian::doccount get_value_freq (Xapian::valueno slot) const`  
*Return the frequency of a given value slot.*
- `std::string get_value_lower_bound (Xapian::valueno slot) const`  
*Get a lower bound on the values stored in the given value slot.*
- `std::string get_value_upper_bound (Xapian::valueno slot) const`  
*Get an upper bound on the values stored in the given value slot.*
- `Xapian::termcount get_doclength_lower_bound () const`  
*Get a lower bound on the length of a document in this DB.*
- `Xapian::termcount get_doclength_upper_bound () const`  
*Get an upper bound on the length of a document in this DB.*
- `Xapian::termcount get_wdf_upper_bound (const std::string &term) const`  
*Get an upper bound on the wdf of term term.*
- `ValueIterator valuestream_begin (Xapian::valueno slot) const`  
*Return an iterator over the value in slot slot for each document.*
- `ValueIteratorEnd_ valuestream_end (Xapian::valueno) const`  
*Return end iterator corresponding to `valuestream_begin()`.*
- `Xapian::termcount get_doclength (Xapian::docid did) const`  
*Get the length of a document.*
- `void keep_alive ()`  
*Send a "keep-alive" to remote databases to stop them timing out.*
- `Xapian::Document get_document (Xapian::docid did) const`  
*Get a document from the database, given its document id.*
- `std::string get_spelling_suggestion (const std::string &word, unsigned max_edit_distance=2) const`  
*Suggest a spelling correction.*
- `Xapian::TermIterator spellings_begin () const`  
*An iterator which returns all the spelling correction targets.*



- [Xapian::TermIterator spellings\\_end](#) () const  
*Corresponding end iterator to [spellings\\_begin\(\)](#).*
- [Xapian::TermIterator synonyms\\_begin](#) (const std::string &term) const  
*An iterator which returns all the synonyms for a given term.*
- [Xapian::TermIterator synonyms\\_end](#) (const std::string &) const  
*Corresponding end iterator to [synonyms\\_begin\(term\)](#).*
- [Xapian::TermIterator synonym\\_keys\\_begin](#) (const std::string &prefix=std::string()) const  
*An iterator which returns all terms which have synonyms.*
- [Xapian::TermIterator synonym\\_keys\\_end](#) (const std::string &=std::string()) const  
*Corresponding end iterator to [synonym\\_keys\\_begin\(prefix\)](#).*
- std::string [get\\_metadata](#) (const std::string &key) const  
*Get the user-specified metadata associated with a given key.*
- [Xapian::TermIterator metadata\\_keys\\_begin](#) (const std::string &prefix=std::string()) const  
*An iterator which returns all user-specified metadata keys.*
- [Xapian::TermIterator metadata\\_keys\\_end](#) (const std::string &=std::string()) const  
*Corresponding end iterator to [metadata\\_keys\\_begin\(\)](#).*
- std::string [get\\_uuid](#) () const  
*Get a UUID for the database.*

### 7.5.1 Detailed Description

This class is used to access a database, or a group of databases.

For searching, this class is used in conjunction with an [Enquire](#) object.

#### Exceptions:

[InvalidArgumentError](#) will be thrown if an invalid argument is supplied, for example, an unknown database type.

[DatabaseOpeningError](#) may be thrown if the database cannot be opened (for example, a required file cannot be found).

[DatabaseVersionError](#) may be thrown if the database is in an unsupported format (for example, created by a newer version of [Xapian](#) which uses an incompatible format).

## 7.5.2 Constructor & Destructor Documentation

### 7.5.2.1 Xapian::Database::Database (const std::string & *path*) [explicit]

Open a [Database](#), automatically determining the database backend to use.

**Parameters:**

*path* directory that the database is stored in.

### 7.5.2.2 virtual Xapian::Database::~~Database () [virtual]

Destroy this handle on the database.

If there are no copies of this object remaining, the database(s) will be closed.

### 7.5.2.3 Xapian::Database::Database (const Database & *other*)

Copying is allowed.

The internals are reference counted, so copying is cheap.

## 7.5.3 Member Function Documentation

### 7.5.3.1 void Xapian::Database::add\_database (const Database & *database*)

Add an existing database (or group of databases) to those accessed by this object.

**Parameters:**

*database* the database(s) to add.

### 7.5.3.2 TermIterator Xapian::Database::allterms\_begin (const std::string & *prefix*) const

An iterator which runs across all terms with a given prefix.

This is functionally similar to getting an iterator with [allterms\\_begin\(\)](#) and then calling `skip_to(prefix)` on that iterator to move to the start of the prefix, but is more convenient (because it detects the end of the prefixed terms), and may be more efficient than simply calling `skip_to()` after opening the iterator, particularly for remote databases.

**Parameters:**

*prefix* The prefix to restrict the returned terms to.

### 7.5.3.3 virtual void Xapian::Database::close () [virtual]

Close the database.

This closes the database and releases all file handles held by the database.

This is a permanent close of the database: calling [reopen\(\)](#) after closing a database will not reopen it, and will raise an exception.

Calling [close\(\)](#) on a database which is already closed has no effect (and doesn't raise an exception).

After this call, calls made to methods of the database (other than [close\(\)](#) or the destructor), or to objects associated with the database will behave in one of the following ways (but which behaviour happens may vary between releases, and between database backends):

- raise a [Xapian::DatabaseError](#) indicating that the database is closed.
- behave exactly as they would have done if the database had not been closed (by using cached data).

To summarise - you should not rely on the exception being raised, or the normal result being available, but if you do get a result, it will be correct.

### 7.5.3.4 Xapian::termcount Xapian::Database::get\_collection\_freq (const std::string & tname) const

Return the total number of occurrences of the given term.

This is the sum of the number of occurrences of the term in each document it indexes: i.e., the sum of the within document frequencies of the term.

#### Parameters:

*tname* The term whose collection frequency is being requested.

### 7.5.3.5 Xapian::termcount Xapian::Database::get\_doclength\_lower\_bound () const

Get a lower bound on the length of a document in this DB.

This bound does not include any zero-length documents.

### 7.5.3.6 Xapian::Document Xapian::Database::get\_document (Xapian::docid did) const

Get a document from the database, given its document id.

This method returns a [Xapian::Document](#) object which provides the information about a document.

**Parameters:**

*did* The document id of the document to retrieve.

**Returns:**

A [Xapian::Document](#) object containing the document data

**Exceptions:**

[Xapian::DocNotFoundError](#) The document specified could not be found in the database.

[Xapian::InvalidArgumentError](#) *did* was 0, which is not a valid document id.

### 7.5.3.7 `std::string Xapian::Database::get_metadata (const std::string & key) const`

Get the user-specified metadata associated with a given key.

User-specified metadata allows you to store arbitrary information in the form of (key,tag) pairs. See [WritableDatabase::set\\_metadata\(\)](#) for more information.

When invoked on a [Xapian::Database](#) object representing multiple databases, currently only the metadata for the first is considered but this behaviour may change in the future.

If there is no piece of metadata associated with the specified key, an empty string is returned (this applies even for backends which don't support metadata).

Empty keys are not valid, and specifying one will cause an exception.

**Parameters:**

*key* The key of the metadata item to access.

**Returns:**

The retrieved metadata item's value.

**Exceptions:**

[Xapian::InvalidArgumentError](#) will be thrown if the key supplied is empty.

### 7.5.3.8 `std::string Xapian::Database::get_spelling_suggestion (const std::string & word, unsigned max_edit_distance = 2) const`

Suggest a spelling correction.

**Parameters:**

*word* The potentially misspelled word.

*max\_edit\_distance* Only consider words which are at most *max\_edit\_distance* edits from *word*. An edit is a character insertion, deletion, or the transposition of two adjacent characters (default is 2).

### 7.5.3.9 `std::string Xapian::Database::get_uuid () const`

Get a UUID for the database.

The UUID will persist for the lifetime of the database.

Replicas (eg, made with the replication protocol, or by copying all the database files) will have the same UUID. However, copies (made with `copydatabase`, or `xapian-compact`) will have different UUIDs.

If the backend does not support UUIDs or this database has no subdatabases, the UUID will be empty.

If this database has multiple sub-databases, the UUID string will contain the UUIDs of all the sub-databases.

### 7.5.3.10 `Xapian::doccount Xapian::Database::get_value_freq (Xapian::valueno slot) const`

Return the frequency of a given value slot.

This is the number of documents which have a (non-empty) value stored in the slot.

#### Parameters:

*slot* The value slot to examine.

#### Exceptions:

*UnimplementedError* The frequency of the value isn't available for this database type.

### 7.5.3.11 `std::string Xapian::Database::get_value_lower_bound (Xapian::valueno slot) const`

Get a lower bound on the values stored in the given value slot.

If there are no values stored in the given value slot, this will return an empty string.

If the lower bound isn't available for the given database type, this will return the lowest possible bound - the empty string.

#### Parameters:

*slot* The value slot to examine.

### 7.5.3.12 `std::string Xapian::Database::get_value_upper_bound (Xapian::valueno slot) const`

Get an upper bound on the values stored in the given value slot.

If there are no values stored in the given value slot, this will return an empty string.

**Parameters:**

*slot* The value slot to examine.

**Exceptions:**

*UnimplementedError* The upper bound of the values isn't available for this database type.

**7.5.3.13 void Xapian::Database::keep\_alive ()**

Send a "keep-alive" to remote databases to stop them timing out.

Has no effect on non-remote databases.

**7.5.3.14 Xapian::TermIterator Xapian::Database::metadata\_keys\_begin (const std::string & *prefix* = std::string()) const**

An iterator which returns all user-specified metadata keys.

When invoked on a [Xapian::Database](#) object representing multiple databases, currently only the metadata for the first is considered but this behaviour may change in the future.

If the backend doesn't support metadata, then this method returns an iterator which compares equal to that returned by [metadata\\_keys\\_end\(\)](#).

**Parameters:**

*prefix* If non-empty, only keys with this prefix are returned.

**Exceptions:**

*Xapian::UnimplementedError* will be thrown if the backend implements user-specified metadata, but doesn't implement iterating its keys (currently this happens for the [InMemory](#) backend).

**7.5.3.15 void Xapian::Database::operator= (const Database & *other*)**

Assignment is allowed.

The internals are reference counted, so assignment is cheap.

**7.5.3.16 PostingIterator Xapian::Database::postlist\_begin (const std::string & *tname*) const**

An iterator pointing to the start of the postlist for a given term.

If the term name is the empty string, the iterator returned will list all the documents in the database. Such an iterator will always return a WDF value of 1, since there is no obvious meaning for this quantity in this case.

### 7.5.3.17 void Xapian::Database::reopen ()

Re-open the database.

This re-opens the database(s) to the latest available version(s). It can be used either to make sure the latest results are returned, or to recover from a [Xapian::DatabaseModifiedError](#).

Calling [reopen\(\)](#) on a database which has been closed (with [close\(\)](#)) will always raise a [Xapian::DatabaseError](#).

### 7.5.3.18 Xapian::TermIterator Xapian::Database::spellings\_begin () const

An iterator which returns all the spelling correction targets.

This returns all the words which are considered as targets for the spelling correction algorithm. The frequency of each word is available as the term frequency of each entry in the returned iterator.

### 7.5.3.19 Xapian::TermIterator Xapian::Database::synonym\_keys\_begin (const std::string & *prefix* = std::string()) const

An iterator which returns all terms which have synonyms.

#### Parameters:

*prefix* If non-empty, only terms with this prefix are returned.

### 7.5.3.20 Xapian::TermIterator Xapian::Database::synonyms\_begin (const std::string & *term*) const

An iterator which returns all the synonyms for a given term.

#### Parameters:

*term* The term to return synonyms for.

### 7.5.3.21 bool Xapian::Database::term\_exists (const std::string & *tname*) const

Check if a given term exists in the database.

Return true if and only if the term exists in the database. This is the same as ([get\\_termfreq\(tname\)](#) != 0), but will often be more efficient.

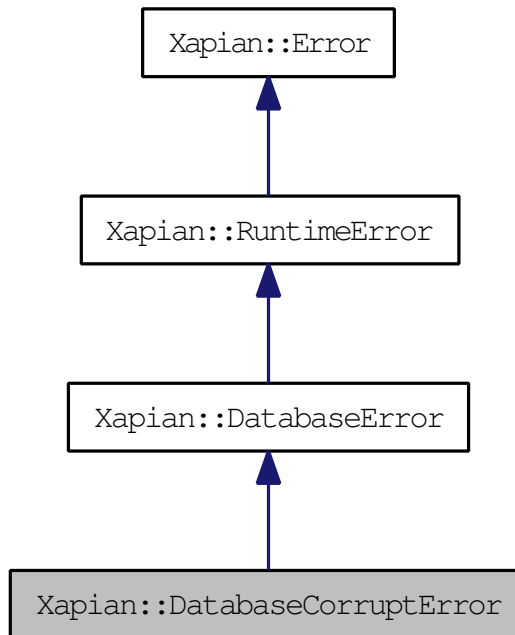
The documentation for this class was generated from the following file:

- [xapian/database.h](#)

## 7.6 Xapian::DatabaseCorruptError Class Reference

[DatabaseCorruptError](#) indicates database corruption was detected.

Inheritance diagram for Xapian::DatabaseCorruptError:



### Public Member Functions

- [DatabaseCorruptError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [DatabaseCorruptError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.6.1 Detailed Description

[DatabaseCorruptError](#) indicates database corruption was detected.



## 7.6.2 Constructor & Destructor Documentation

**7.6.2.1 Xapian::DatabaseCorruptError::DatabaseCorruptError (const std::string & *msg\_*, const std::string & *context\_* = std::string(), int *errno\_* = 0) [inline, explicit]**

General purpose constructor which allows setting errno.

**7.6.2.2 Xapian::DatabaseCorruptError::DatabaseCorruptError (const std::string & *msg\_*, int *errno\_*) [inline]**

Construct from message and errno value.

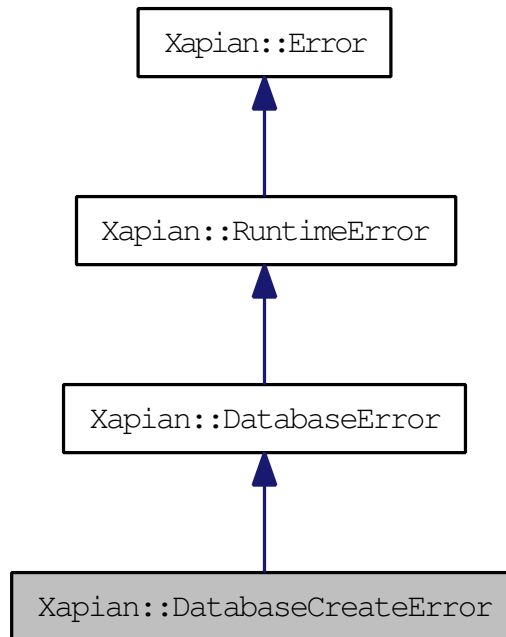
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.7 Xapian::DatabaseCreateError Class Reference

[DatabaseCreateError](#) indicates a failure to create a database.

Inheritance diagram for Xapian::DatabaseCreateError:



### Public Member Functions

- [DatabaseCreateError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [DatabaseCreateError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.7.1 Detailed Description

[DatabaseCreateError](#) indicates a failure to create a database.

## 7.7.2 Constructor & Destructor Documentation

**7.7.2.1 Xapian::DatabaseCreateError::DatabaseCreateError (const std::string & msg\_, const std::string & context\_ = std::string(), int errno\_ = 0) [inline, explicit]**

General purpose constructor which allows setting errno.

**7.7.2.2 Xapian::DatabaseCreateError::DatabaseCreateError (const std::string & msg\_, int errno\_) [inline]**

Construct from message and errno value.

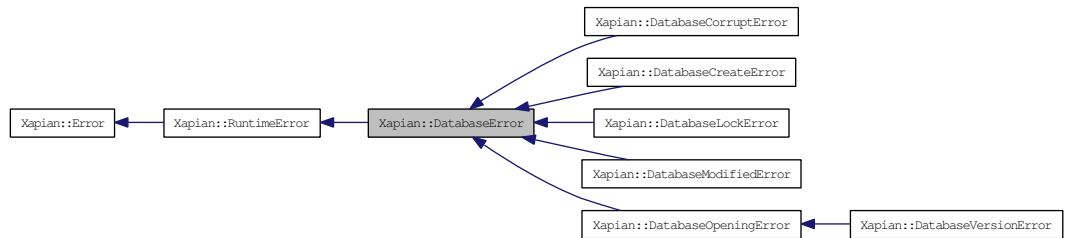
The documentation for this class was generated from the following file:

- xapian/[error.h](#)

## 7.8 Xapian::DatabaseError Class Reference

[DatabaseError](#) indicates some sort of database related error.

Inheritance diagram for Xapian::DatabaseError:



### Public Member Functions

- [DatabaseError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_=0)  
*General purpose constructor which allows setting errno.*
- [DatabaseError](#) (const std::string &msg\_, int errno\_)  
*Construct from message and errno value.*

### 7.8.1 Detailed Description

[DatabaseError](#) indicates some sort of database related error.

### 7.8.2 Constructor & Destructor Documentation

**7.8.2.1 Xapian::DatabaseError::DatabaseError (const std::string & msg\_, const std::string & context\_ = std::string(), int errno\_ = 0) [inline, explicit]**

General purpose constructor which allows setting errno.

**7.8.2.2 Xapian::DatabaseError::DatabaseError (const std::string & msg\_, int errno\_) [inline]**

Construct from message and errno value.

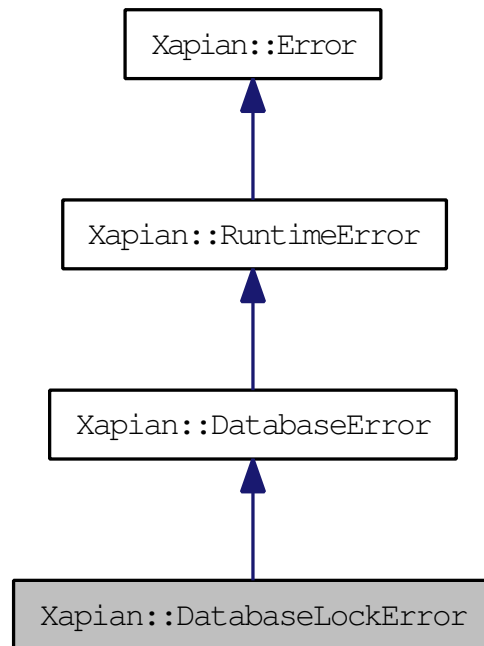
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.9 Xapian::DatabaseLockError Class Reference

[DatabaseLockError](#) indicates failure to lock a database.

Inheritance diagram for Xapian::DatabaseLockError:



### Public Member Functions

- [DatabaseLockError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [DatabaseLockError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.9.1 Detailed Description

[DatabaseLockError](#) indicates failure to lock a database.

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 `Xapian::DatabaseLockError::DatabaseLockError (const std::string & msg_, const std::string & context_ = std::string(), int errno_ = 0)` `[inline, explicit]`

General purpose constructor which allows setting errno.

### 7.9.2.2 `Xapian::DatabaseLockError::DatabaseLockError (const std::string & msg_, int errno_)` `[inline]`

Construct from message and errno value.

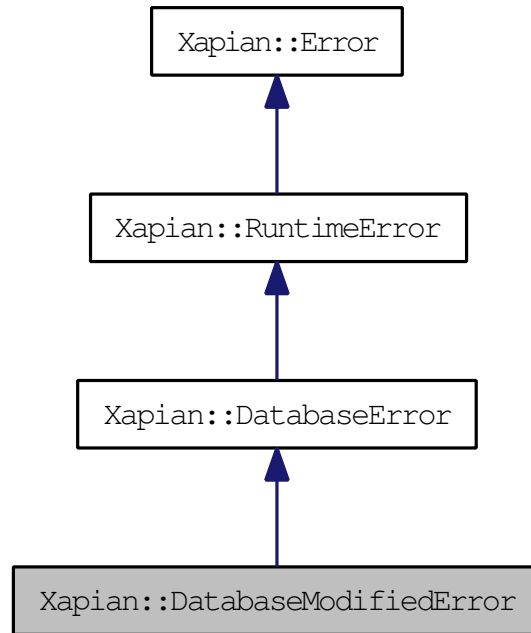
The documentation for this class was generated from the following file:

- `xapian/error.h`

## 7.10 Xapian::DatabaseModifiedError Class Reference

[DatabaseModifiedError](#) indicates a database was modified.

Inheritance diagram for Xapian::DatabaseModifiedError:



### Public Member Functions

- [DatabaseModifiedError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [DatabaseModifiedError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.10.1 Detailed Description

[DatabaseModifiedError](#) indicates a database was modified.

To recover after catching this error, you need to call [Xapian::Database::reopen\(\)](#) on the [Database](#) and repeat the operation which failed.

## 7.10.2 Constructor & Destructor Documentation

### 7.10.2.1 Xapian::DatabaseModifiedError::DatabaseModifiedError (const std::string & msg\_, const std::string & context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

### 7.10.2.2 Xapian::DatabaseModifiedError::DatabaseModifiedError (const std::string & msg\_, int errno\_) [inline]

Construct from message and errno value.

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

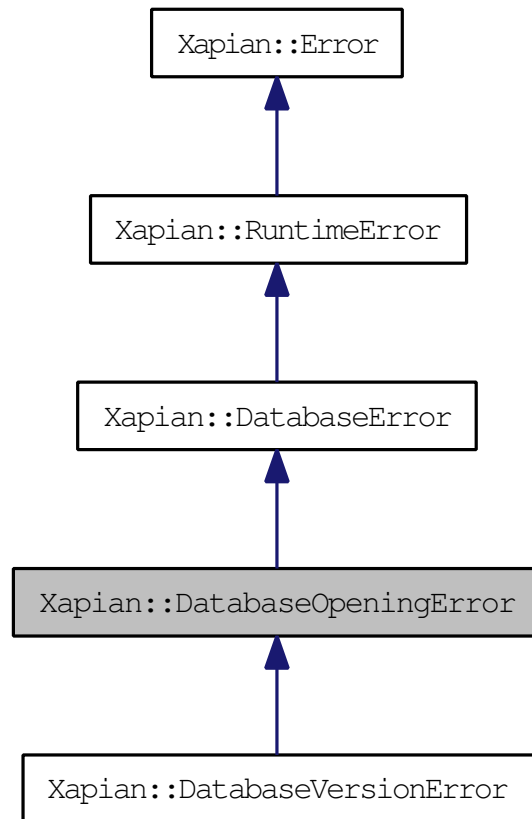
- xapian/[error.h](#)



## 7.11 Xapian::DatabaseOpeningError Class Reference

[DatabaseOpeningError](#) indicates failure to open a database.

Inheritance diagram for Xapian::DatabaseOpeningError:



### Public Member Functions

- [DatabaseOpeningError](#) (const std::string &msg\_, const std::string &context\_  
=std::string(), int errno\_=0)  
*General purpose constructor which allows setting errno.*
- [DatabaseOpeningError](#) (const std::string &msg\_, int errno\_)  
*Construct from message and errno value.*

### 7.11.1 Detailed Description

[DatabaseOpeningError](#) indicates failure to open a database.

## 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 Xapian::DatabaseOpeningError::DatabaseOpeningError (const std::string & msg\_, const std::string & context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

### 7.11.2.2 Xapian::DatabaseOpeningError::DatabaseOpeningError (const std::string & msg\_, int errno\_) [inline]

Construct from message and errno value.

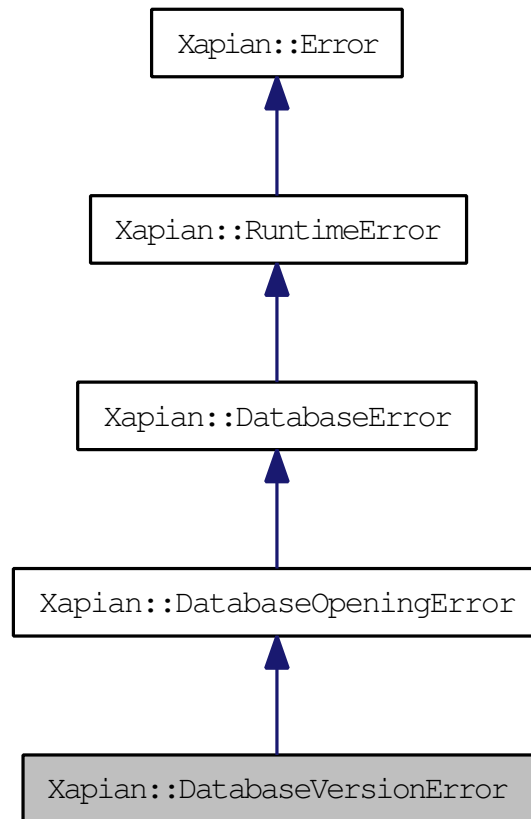
The documentation for this class was generated from the following file:

- xapian/[error.h](#)

## 7.12 Xapian::DatabaseVersionError Class Reference

[DatabaseVersionError](#) indicates that a database is in an unsupported format.

Inheritance diagram for Xapian::DatabaseVersionError:



### Public Member Functions

- [DatabaseVersionError](#) (const std::string &msg\_, const std::string &context\_  
=std::string(), int errno\_=0)  
*General purpose constructor which allows setting errno.*
- [DatabaseVersionError](#) (const std::string &msg\_, int errno\_)  
*Construct from message and errno value.*

### 7.12.1 Detailed Description

[DatabaseVersionError](#) indicates that a database is in an unsupported format.

From time to time, new versions of [Xapian](#) will require the database format to be changed, to allow new information to be stored or new optimisations to be performed. Backwards compatibility will sometimes be maintained, so that new versions of [Xapian](#) can open old databases, but in some cases [Xapian](#) will be unable to open a database because it is in too old (or new) a format. This can be resolved either by upgrading or downgrading the version of [Xapian](#) in use, or by rebuilding the database from scratch with the current version of [Xapian](#).

## 7.12.2 Constructor & Destructor Documentation

### 7.12.2.1 `Xapian::DatabaseVersionError::DatabaseVersionError (const std::string & msg_, const std::string & context_ = std::string(), int errno_ = 0) [inline, explicit]`

General purpose constructor which allows setting errno.

### 7.12.2.2 `Xapian::DatabaseVersionError::DatabaseVersionError (const std::string & msg_, int errno_) [inline]`

Construct from message and errno value.

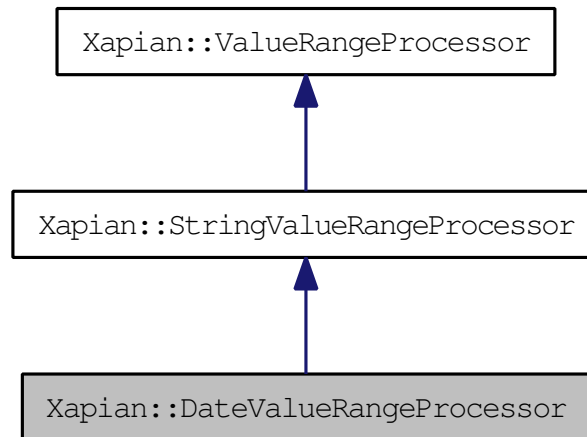
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.13 Xapian::DateValueRangeProcessor Class Reference

Handle a date range.

Inheritance diagram for Xapian::DateValueRangeProcessor:



### Public Member Functions

- [DateValueRangeProcessor](#) ([Xapian::valueno](#) slot\_, bool prefer\_mdy\_=false, int epoch\_year\_=1970)

*Constructor.*

- [DateValueRangeProcessor](#) ([Xapian::valueno](#) slot\_, const std::string &str\_, bool prefix\_=true, bool prefer\_mdy\_=false, int epoch\_year\_=1970)

*Constructor.*

- [Xapian::valueno operator\(\)](#) (std::string &begin, std::string &end)

*Check for a valid range of this type.*

#### 7.13.1 Detailed Description

Handle a date range.

Begin and end must be dates in a recognised format.

## 7.13.2 Constructor & Destructor Documentation

**7.13.2.1 Xapian::DateValueRangeProcessor::DateValueRangeProcessor**  
**(Xapian::valueno *slot\_*, bool *prefer\_mdy\_* = false, int *epoch\_year\_* = 1970) [inline]**

Constructor.

### Parameters:

- slot\_* The value number to return from operator().
- prefer\_mdy\_* Should ambiguous dates be interpreted as month/day/year rather than day/month/year? (default: false)
- epoch\_year\_* Year to use as the epoch for dates with 2 digit years (default: 1970, so 1/1/69 is 2069 while 1/1/70 is 1970).

**7.13.2.2 Xapian::DateValueRangeProcessor::DateValueRangeProcessor**  
**(Xapian::valueno *slot\_*, const std::string & *str\_*, bool *prefix\_* = true, bool *prefer\_mdy\_* = false, int *epoch\_year\_* = 1970) [inline]**

Constructor.

### Parameters:

- slot\_* The value number to return from operator().
- str\_* A string to look for to recognise values as belonging to this date range.
- prefix\_* Whether to look for the string at the start or end of the values. If true, the string is a prefix; if false, the string is a suffix (default: true).
- prefer\_mdy\_* Should ambiguous dates be interpreted as month/day/year rather than day/month/year? (default: false)
- epoch\_year\_* Year to use as the epoch for dates with 2 digit years (default: 1970, so 1/1/69 is 2069 while 1/1/70 is 1970).

The string supplied in *str\_* is used by *operator()* to decide whether the pair of strings supplied to it constitute a valid range. If *prefix\_* is true, the first value in a range must begin with *str\_* (and the second value may optionally begin with *str\_*); if *prefix\_* is false, the second value in a range must end with *str\_* (and the first value may optionally end with *str\_*).

If *str\_* is empty, the setting of *prefix\_* is irrelevant, and no special strings are required at the start or end of the strings defining the range.

The remainder of both strings defining the endpoints must be valid dates.

For example, if *str\_* is "created:" and *prefix\_* is true, and the range processor has been added to the queryparser, the queryparser will accept "created:1/1/2000..31/12/2001".

### 7.13.3 Member Function Documentation

#### 7.13.3.1 Xapian::valueno Xapian::DateValueRangeProcessor::operator() (std::string & *begin*, std::string & *end*) [virtual]

Check for a valid range of this type.

If BEGIN..END is a sensible date range, this method returns the value number of range filter on. Otherwise it returns [Xapian::BAD\\_VALUENO](#).

Reimplemented from [Xapian::StringValueRangeProcessor](#).

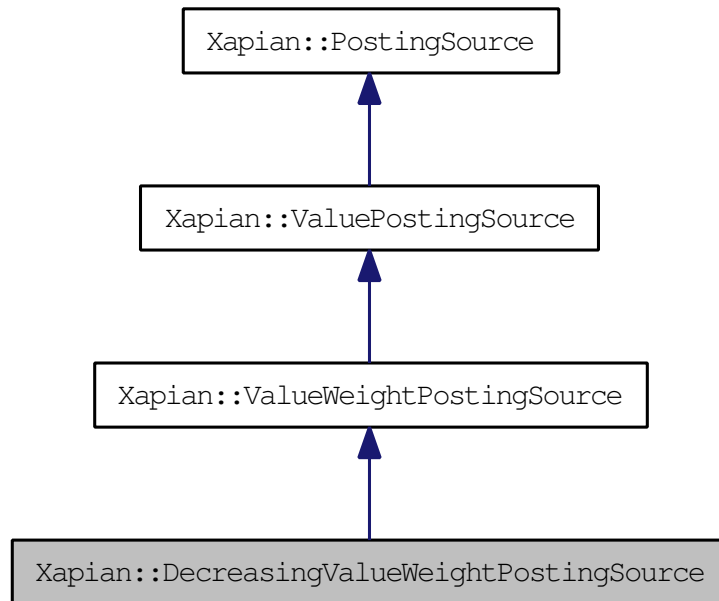
The documentation for this class was generated from the following file:

- [xapian/queryparser.h](#)

## 7.14 Xapian::DecreasingValueWeightPostingSource Class Reference

Read weights from a value which is known to decrease as docid increases.

Inheritance diagram for Xapian::DecreasingValueWeightPostingSource:



### Public Member Functions

- [Xapian::weight](#) [get\\_weight](#) () const  
*Return the weight contribution for the current document.*
- [DecreasingValueWeightPostingSource](#) \* [clone](#) () const  
*Clone the posting source.*
- std::string [name](#) () const  
*Name of the posting source class.*
- std::string [serialise](#) () const  
*Serialise object parameters into a string.*
- [DecreasingValueWeightPostingSource](#) \* [unserialise](#) (const std::string &s)  
const  
*Create object given string serialisation returned by [serialise\(\)](#).*
- void [init](#) (const [Xapian::Database](#) &db\_)



Set this [PostingSource](#) to the start of the list of postings.

- void [next](#) ([Xapian::weight](#) min\_wt)  
*Advance the current position to the next matching document.*
- void [skip\\_to](#) ([Xapian::docid](#) min\_docid, [Xapian::weight](#) min\_wt)  
*Advance to the specified docid.*
- bool [check](#) ([Xapian::docid](#) min\_docid, [Xapian::weight](#) min\_wt)  
*Check if the specified docid occurs.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

## Protected Member Functions

- void [skip\\_if\\_in\\_range](#) ([Xapian::weight](#) min\_wt)  
*Skip the iterator forward if in the decreasing range, and weight is low.*

## Protected Attributes

- bool [items\\_at\\_end](#)  
*Flag, set to true if there are docs after the end of the range.*

### 7.14.1 Detailed Description

Read weights from a value which is known to decrease as docid increases.

This posting source can be used, like [ValueWeightPostingSource](#), to add a weight contribution to a query based on the values stored in a slot. The values in the slot must be serialised as by [sortable\\_serialise\(\)](#).

However, this posting source is additionally given a range of document IDs, within which the weight is known to be decreasing. ie, for all documents with ids A and B within this range (including the endpoints), where A is less than B, the weight of A is less than or equal to the weight of B. This can allow the posting source to skip to the end of the range quickly if insufficient weight is left in the posting source for a particular source.

By default, the range is assumed to cover all document IDs.

The ordering property can be arranged at index time, or by sorting an indexed database to produce a new, sorted, database.

## 7.14.2 Member Function Documentation

### 7.14.2.1 `bool Xapian::DecreasingValueWeightPostingSource::check` `(Xapian::docid did, Xapian::weight min_wt)` `[virtual]`

Check if the specified docid occurs.

The caller is required to ensure that the specified document id *did* actually exists in the database. If it does, it must move to that document id, and return true. If it does not, it may either:

- return true, having moved to a definite position (including "at\_end"), which must be the same position as `skip_to()` would have moved to.

or

- return false, having moved to an "indeterminate" position, such that a subsequent call to `next()` or `skip_to()` will move to the next matching position after *did*.

Generally, this method should act like `skip_to()` and return true if that can be done at little extra cost.

Otherwise it should simply check if a particular docid is present, returning true if it is, and false if it isn't.

The default implementation calls `skip_to()` and always returns true.

`Xapian` will always call `init()` on a `PostingSource` before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the single subdatabase relevant to this posting source. See the `init()` method for details.

Reimplemented from `Xapian::ValuePostingSource`.

### 7.14.2.2 `DecreasingValueWeightPostingSource*` `Xapian::DecreasingValueWeightPostingSource::clone ()` `const` `[virtual]`

Clone the posting source.

The clone should inherit the configuration of the parent, but need not inherit the state. ie, the clone does not need to be in the same iteration position as the original: the matcher will always call `init()` on the clone before attempting to move the iterator, or read the information about the current position of the iterator.

This may return NULL to indicate that cloning is not supported. In this case, the `PostingSource` may only be used with a single-database search.

The default implementation returns NULL.

Note that the returned object will be deallocated by `Xapian` after use with "delete". It must therefore have been allocated with "new".

Reimplemented from `Xapian::ValueWeightPostingSource`.

### 7.14.2.3 `std::string Xapian::DecreasingValueWeightPostingSource::get_description() const` [virtual]

Return a string describing this object.

This default implementation returns a generic answer. This default is provided to avoid forcing those deriving their own [PostingSource](#) subclass from having to implement this (they may not care what [get\\_description\(\)](#) gives for their subclass).

Reimplemented from [Xapian::ValueWeightPostingSource](#).

### 7.14.2.4 `Xapian::weight Xapian::DecreasingValueWeightPostingSource::get_weight() const` [virtual]

Return the weight contribution for the current document.

This default implementation always returns 0, for convenience when implementing "weight-less" [PostingSource](#) subclasses.

This method may assume that it will only be called when there is a "current document". In detail: [Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time. It will also only call this if the [PostingSource](#) reports that it is pointing to a valid document (ie, it will not call it before calling at least one of [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#), and will ensure that the [PostingSource](#) is not at the end by calling [at\\_end\(\)](#)).

Reimplemented from [Xapian::ValueWeightPostingSource](#).

### 7.14.2.5 `void Xapian::DecreasingValueWeightPostingSource::init (const Xapian::Database & db)` [virtual]

Set this [PostingSource](#) to the start of the list of postings.

This is called automatically by the matcher prior to each query being processed.

If a [PostingSource](#) is used for multiple searches, [init\(\)](#) will therefore be called multiple times, and must handle this by using the database passed in the most recent call.

#### Parameters:

*db* The database which the [PostingSource](#) should iterate through.

Note: the database supplied to this method must not be modified: in particular, the [reopen\(\)](#) method should not be called on it.

Note: in the case of a multi-database search, a separate [PostingSource](#) will be used for each database (the separate [PostingSources](#) will be obtained using [clone\(\)](#)), and each [PostingSource](#) will be passed one of the sub-databases as the *db* parameter here. The *db* parameter will therefore always refer to a single database. All docids passed to, or returned from, the [PostingSource](#) refer to docids in that single database, rather than in the multi-database.

Reimplemented from [Xapian::ValueWeightPostingSource](#).

#### 7.14.2.6 `std::string Xapian::DecreasingValueWeightPostingSource::name ()` `const` [virtual]

Name of the posting source class.

This is used when serialising and unserialising posting sources; for example, for performing remote searches.

If the subclass is in a C++ namespace, the namespace should be included in the name, using "::" as a separator. For example, for a [PostingSource](#) subclass called "FooPostingSource" in the "Xapian" namespace the result of this call should be "Xapian::FooPostingSource".

This should only be implemented if [serialise\(\)](#) and [unserialise\(\)](#) are also implemented. The default implementation returns an empty string.

If this returns an empty string, [Xapian](#) will assume that [serialise\(\)](#) and [unserialise\(\)](#) are not implemented.

Reimplemented from [Xapian::ValueWeightPostingSource](#).

#### 7.14.2.7 `void Xapian::DecreasingValueWeightPostingSource::next` `(Xapian::weight min_wt)` [virtual]

Advance the current position to the next matching document.

The [PostingSource](#) starts before the first entry in the list, so [next\(\)](#) must be called before any methods which need the context of the current position.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

##### Parameters:

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Reimplemented from [Xapian::ValuePostingSource](#).

#### 7.14.2.8 `std::string Xapian::DecreasingValueWeightPostingSource::serialise ()` `const` [virtual]

Serialise object parameters into a string.

The serialised parameters should represent the configuration of the posting source, but need not (indeed, should not) represent the current iteration state.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::ValueWeightPostingSource](#).

#### 7.14.2.9 void Xapian::DecreasingValueWeightPostingSource::skip\_to (Xapian::docid *did*, Xapian::weight *min\_wt*) [virtual]

Advance to the specified docid.

If the specified docid isn't in the list, position ourselves on the first document after it (or `at_end()` if no greater docids are present).

If the current position is already the specified docid, this method will leave the position unmodified.

If the specified docid is earlier than the current position, the behaviour is unspecified. A sensible behaviour would be to leave the current position unmodified, but it is also reasonable to move to the specified docid.

The default implementation calls `next()` repeatedly, which works but `skip_to()` can often be implemented much more efficiently.

[Xapian](#) will always call `init()` on a [PostingSource](#) before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the single subdatabase relevant to this posting source. See the `init()` method for details.

##### Parameters:

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Reimplemented from [Xapian::ValuePostingSource](#).

#### 7.14.2.10 DecreasingValueWeightPostingSource\* Xapian::DecreasingValueWeightPostingSource::unserialise (const std::string & *s*) const [virtual]

Create object given string serialisation returned by `serialise()`.

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

##### Parameters:

*s* A serialised instance of this [PostingSource](#) subclass.

Reimplemented from [Xapian::ValueWeightPostingSource](#).

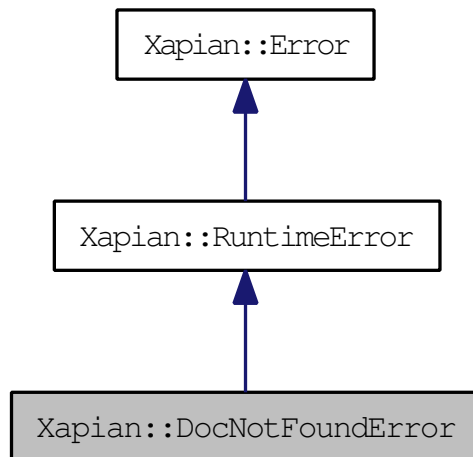
The documentation for this class was generated from the following file:

- [xapian/postingsource.h](#)

## 7.15 Xapian::DocNotFoundError Class Reference

Indicates an attempt to access a document not present in the database.

Inheritance diagram for Xapian::DocNotFoundError:



### Public Member Functions

- [DocNotFoundError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [DocNotFoundError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.15.1 Detailed Description

Indicates an attempt to access a document not present in the database.

### 7.15.2 Constructor & Destructor Documentation

- #### 7.15.2.1 Xapian::DocNotFoundError::DocNotFoundError (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

### 7.15.2.2 Xapian::DocNotFoundError::DocNotFoundError (const std::string & msg\_, int *errno*\_)

[inline]

Construct from message and errno value.

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

- [xapian/error.h](#)

## 7.16 Xapian::Document Class Reference

A handle representing a document in a [Xapian](#) database.

### Public Member Functions

- [Document](#) (const [Document](#) &other)  
*Copying is allowed.*
- void [operator=](#) (const [Document](#) &other)  
*Assignment is allowed.*
- [Document](#) ()  
*Make a new empty [Document](#).*
- [~Document](#) ()  
*Destructor.*
- std::string [get\\_value](#) ([Xapian::valueno](#) slot) const  
*Get value by number.*
- void [add\\_value](#) ([Xapian::valueno](#) slot, const std::string &value)  
*Add a new value.*
- void [remove\\_value](#) ([Xapian::valueno](#) slot)  
*Remove any value with the given number.*
- void [clear\\_values](#) ()  
*Remove all values associated with the document.*
- std::string [get\\_data](#) () const  
*Get data stored in the document.*
- void [set\\_data](#) (const std::string &data)  
*Set data stored in the document.*
- void [add\\_posting](#) (const std::string &tname, [Xapian::termpos](#) tpos, [Xapian::termcount](#) wdfinc=1)  
*Add an occurrence of a term at a particular position.*
- void [add\\_term](#) (const std::string &tname, [Xapian::termcount](#) wdfinc=1)  
*Add a term to the document, without positional information.*
- void [add\\_boolean\\_term](#) (const std::string &term)  
*Add a boolean filter term to the document.*



- void [remove\\_posting](#) (const std::string &tname, [Xapian::termpos](#) tpos, [Xapian::termcount](#) wdfdec=1)  
*Remove a posting of a term from the document.*
- void [remove\\_term](#) (const std::string &tname)  
*Remove a term and all postings associated with it.*
- void [clear\\_terms](#) ()  
*Remove all terms (and postings) from the document.*
- [Xapian::termcount](#) [termlist\\_count](#) () const  
*The length of the termlist - i.e.*
- [TermIterator](#) [termlist\\_begin](#) () const  
*Iterator for the terms in this document.*
- [TermIterator](#) [termlist\\_end](#) () const  
*Equivalent end iterator for [termlist\\_begin\(\)](#).*
- [Xapian::termcount](#) [values\\_count](#) () const  
*Count the values in this document.*
- [ValueIterator](#) [values\\_begin](#) () const  
*Iterator for the values in this document.*
- [ValueIteratorEnd](#) [\\_values\\_end](#) () const  
*Equivalent end iterator for [values\\_begin\(\)](#).*
- [docid](#) [get\\_docid](#) () const  
*Get the document id which is associated with this document (if any).*
- std::string [serialise](#) () const  
*Serialise document into a string.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

## Static Public Member Functions

- static [Document](#) [unserialise](#) (const std::string &s)  
*Unserialise a document from a string produced by [serialise\(\)](#).*

### 7.16.1 Detailed Description

A handle representing a document in a [Xapian](#) database.

The [Document](#) class fetches information from the database lazily. Usually this behaviour isn't visible to users (except for the speed benefits), but if the document in the database is modified or deleted, then preexisting [Document](#) objects may return the old or new versions of data (or throw [Xapian::DocNotFoundError](#) in the case of deletion).

Since [Database](#) objects work on a snapshot of the database's state, the situation above can only happen with a [WritableDatabase](#) object, or if you call [Database::reopen\(\)](#) on a [Database](#) object.

We recommend you avoid designs where this behaviour is an issue, but if you need a way to make a non-lazy version of a [Document](#) object, you can do this like so:

```
doc = Xapian::Document::unserialise(doc.serialise());
```

### 7.16.2 Constructor & Destructor Documentation

#### 7.16.2.1 [Xapian::Document::Document](#) (const [Document](#) & *other*)

Copying is allowed.

The internals are reference counted, so copying is cheap.

### 7.16.3 Member Function Documentation

#### 7.16.3.1 [void Xapian::Document::add\\_boolean\\_term](#) (const [std::string](#) & *term*) [[inline](#)]

Add a boolean filter term to the document.

This method adds *term* to the document with wdf of 0 - this is generally what you want for a term used for boolean filtering as the wdf of such terms is ignored, and it doesn't make sense for them to contribute to the document's length.

If the specified term already indexes this document, this method has no effect.

It is exactly the same as `add_term(term, 0)`.

This method was added in [Xapian](#) 1.0.18.

#### Parameters:

*term* The term to add.

#### 7.16.3.2 [void Xapian::Document::add\\_posting](#) (const [std::string](#) & *tname*, [Xapian::termpos](#) *tpos*, [Xapian::termcount](#) *wdfinc* = 1)

Add an occurrence of a term at a particular position.

Multiple occurrences of the term at the same position are represented only once in the positional information, but do increase the wdf.

If the term is not already in the document, it will be added to it.

**Parameters:**

- tname* The name of the term.
- tpos* The position of the term.
- wdfinc* The increment that will be applied to the wdf for this term.

**7.16.3.3 void Xapian::Document::add\_term (const std::string & *tname*, Xapian::termcount *wdfinc* = 1)**

Add a term to the document, without positional information.

Any existing positional information for the term will be left unmodified.

**Parameters:**

- tname* The name of the term.
- wdfinc* The increment that will be applied to the wdf for this term (default: 1).

**7.16.3.4 void Xapian::Document::add\_value (Xapian::valueno *slot*, const std::string & *value*)**

Add a new value.

The new value will replace any existing value with the same number (or if the new value is empty, it will remove any existing value with the same number).

**7.16.3.5 std::string Xapian::Document::get\_data () const**

Get data stored in the document.

This is a potentially expensive operation, and shouldn't normally be used in a match decider functor. Put data for use by match deciders in a value instead.

**7.16.3.6 docid Xapian::Document::get\_docid () const**

Get the document id which is associated with this document (if any).

NB If multiple databases are being searched together, then this will be the document id in the individual database, not the merged database!

**Returns:**

If this document came from a database, return the document id in that database. Otherwise, return 0 (in [Xapian 1.0.22/1.2.4](#) or later; prior to this the returned value was uninitialised).

### 7.16.3.7 `std::string Xapian::Document::get_value (Xapian::value slot) const`

Get value by number.

Returns an empty string if no value with the given number is present in the document.

#### Parameters:

*slot* The number of the value.

### 7.16.3.8 `void Xapian::Document::operator= (const Document & other)`

Assignment is allowed.

The internals are reference counted, so assignment is cheap.

### 7.16.3.9 `void Xapian::Document::remove_posting (const std::string & tname, Xapian::termpos tpos, Xapian::termcount wdfdec = 1)`

Remove a posting of a term from the document.

Note that the term will still index the document even if all occurrences are removed.

To remove a term from a document completely, use [remove\\_term\(\)](#).

#### Parameters:

*tname* The name of the term.

*tpos* The position of the term.

*wdfdec* The decrement that will be applied to the wdf when removing this posting.  
The wdf will not go below the value of 0.

#### Exceptions:

[\*Xapian::InvalidArgumentError\*](#) will be thrown if the term is not at the position specified in the position list for this term in this document.

[\*Xapian::InvalidArgumentError\*](#) will be thrown if the term is not in the document

### 7.16.3.10 `void Xapian::Document::remove_term (const std::string & tname)`

Remove a term and all postings associated with it.

#### Parameters:

*tname* The name of the term.

#### Exceptions:

[\*Xapian::InvalidArgumentError\*](#) will be thrown if the term is not in the document

**7.16.3.11 std::string Xapian::Document::serialise () const**

Serialise document into a string.

The document representation may change between [Xapian](#) releases: even between minor versions. However, it is guaranteed not to change if the remote database protocol has not changed between releases.

**7.16.3.12 Xapian::termcount Xapian::Document::termlist\_count () const**

The length of the termlist - i.e.

the number of different terms which index this document.

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

- [xapian/document.h](#)

## 7.17 Xapian::Enquire Class Reference

This class provides an interface to the information retrieval system for the purpose of searching.

### Public Member Functions

- [Enquire](#) (const [Enquire](#) &other)  
*Copying is allowed (and is cheap).*
- void [operator=](#) (const [Enquire](#) &other)  
*Assignment is allowed (and is cheap).*
- [Enquire](#) (const [Database](#) &database, [ErrorHandler](#) \*errorhandler\_=0)  
*Create a [Xapian::Enquire](#) object.*
- [~Enquire](#) ()  
*Close the [Xapian::Enquire](#) object.*
- void [set\\_query](#) (const [Xapian::Query](#) &query, [Xapian::termcount](#) qlen=0)  
*Set the query to run.*
- const [Xapian::Query](#) & [get\\_query](#) () const  
*Get the query which has been set.*
- void [add\\_matchspy](#) ([MatchSpy](#) \*spy)  
*Add a matchspy.*
- void [clear\\_matchspies](#) ()  
*Remove all the matchspies.*
- void [set\\_weighting\\_scheme](#) (const [Weight](#) &weight\_)  
*Set the weighting scheme to use for queries.*
- void [set\\_collapse\\_key](#) ([Xapian::valueno](#) collapse\_key, [Xapian::doccount](#) collapse\_max=1)  
*Set the collapse key to use for queries.*
- void [set\\_docid\\_order](#) (docid\_order order)  
*Set the direction in which documents are ordered by document id in the returned [MSet](#).*
- void [set\\_cutoff](#) ([Xapian::percent](#) percent\_cutoff, [Xapian::weight](#) weight\_cutoff=0)  
*Set the percentage and/or weight cutoffs.*
- void [set\\_sort\\_by\\_relevance](#) ()

*Set the sorting to be by relevance only.*

- void [set\\_sort\\_by\\_value](#) ([Xapian::valueno](#) sort\_key, bool reverse)  
*Set the sorting to be by value only.*
- void [set\\_sort\\_by\\_key](#) ([Xapian::KeyMaker](#) \*sorter, bool reverse)  
*Set the sorting to be by key generated from values only.*
- void [set\\_sort\\_by\\_value\\_then\\_relevance](#) ([Xapian::valueno](#) sort\_key, bool reverse)  
*Set the sorting to be by value, then by relevance for documents with the same value.*
- void [set\\_sort\\_by\\_key\\_then\\_relevance](#) ([Xapian::KeyMaker](#) \*sorter, bool reverse)  
*Set the sorting to be by keys generated from values, then by relevance for documents with identical keys.*
- void [set\\_sort\\_by\\_relevance\\_then\\_value](#) ([Xapian::valueno](#) sort\_key, bool reverse)  
*Set the sorting to be by relevance then value.*
- void [set\\_sort\\_by\\_relevance\\_then\\_key](#) ([Xapian::KeyMaker](#) \*sorter, bool reverse)  
*Set the sorting to be by relevance, then by keys generated from values.*
- [ESet](#) [get\\_eset](#) ([Xapian::termcount](#) maxitems, const [RSet](#) &omrset, int flags=0, double k=1.0, const [Xapian::ExpandDecider](#) \*edecider=0) const  
*Get the expand set for the given rset.*
- [ESet](#) [get\\_eset](#) ([Xapian::termcount](#) maxitems, const [RSet](#) &omrset, const [Xapian::ExpandDecider](#) \*edecider) const  
*Get the expand set for the given rset.*
- [ESet](#) [get\\_eset](#) ([Xapian::termcount](#) maxitems, const [RSet](#) &omrset, int flags, double k, const [Xapian::ExpandDecider](#) \*edecider, [Xapian::weight](#) min\_wt) const  
*Get the expand set for the given rset.*
- [TermIterator](#) [get\\_matching\\_terms\\_begin](#) ([Xapian::docid](#) did) const  
*Get terms which match a given document, by document id.*
- [TermIterator](#) [get\\_matching\\_terms\\_end](#) ([Xapian::docid](#)) const  
*End iterator corresponding to [get\\_matching\\_terms\\_begin\(\)](#).*
- [TermIterator](#) [get\\_matching\\_terms\\_begin](#) (const [MSetIterator](#) &it) const  
*Get terms which match a given document, by match set item.*
- [TermIterator](#) [get\\_matching\\_terms\\_end](#) (const [MSetIterator](#) &) const

*End iterator corresponding to [get\\_matching\\_terms\\_begin\(\)](#).*

- `std::string get\_description () const`  
*Return a string describing this object.*
- `MSet get\_mset (Xapian::doccount first, Xapian::doccount maxitems, Xapian::doccount checkatleast=0, const RSet *omrset=0, const MatchDecider *mdecider=0) const`  
*Get (a portion of) the match set for the current query.*
- `MSet get\_mset (Xapian::doccount first, Xapian::doccount maxitems, Xapian::doccount checkatleast, const RSet *omrset, const MatchDecider *mdecider, const MatchDecider *matchspy) const`  
*Get (a portion of) the match set for the current query.*
- `MSet get\_mset (Xapian::doccount first, Xapian::doccount maxitems, const RSet *omrset, const MatchDecider *mdecider=0) const`  
*Get (a portion of) the match set for the current query.*

### 7.17.1 Detailed Description

This class provides an interface to the information retrieval system for the purpose of searching.

Databases are usually opened lazily, so exceptions may not be thrown where you would expect them to be. You should catch [Xapian::Error](#) exceptions when calling any method in [Xapian::Enquire](#).

#### Exceptions:

[Xapian::InvalidArgumentError](#) will be thrown if an invalid argument is supplied, for example, an unknown database type.

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 [Xapian::Enquire::Enquire](#) (const Database & *database*, ErrorHandler \* *errorhandler\_* = 0) [explicit]

Create a [Xapian::Enquire](#) object.

This specification cannot be changed once the [Xapian::Enquire](#) is opened: you must create a new [Xapian::Enquire](#) object to access a different database, or set of databases.

The database supplied must have been initialised (ie, must not be the result of calling the [Database::Database\(\)](#) constructor). If you need to handle a situation where you have no index gracefully, a database created with [InMemory::open\(\)](#) can be passed here, which represents a completely empty database.



**Parameters:**

*database* Specification of the database or databases to use.

*errorhandler\_* A pointer to the error handler to use. Ownership of the object pointed to is not assumed by the [Xapian::Enquire](#) object - the user should delete the [Xapian::ErrorHandler](#) object after the [Xapian::Enquire](#) object is deleted. To use no error handler, this parameter should be 0.

**Exceptions:**

[Xapian::InvalidArgumentError](#) will be thrown if an initialised [Database](#) object is supplied.

**7.17.3 Member Function Documentation****7.17.3.1 void Xapian::Enquire::add\_matchspy (MatchSpy \* *spy*)**

Add a matchspy.

This matchspy will be called with some of the documents which match the query, during the match process. Exactly which of the matching documents are passed to it depends on exactly when certain optimisations occur during the match process, but it can be controlled to some extent by setting the *checkatleast* parameter to [get\\_mset\(\)](#).

In particular, if there are enough matching documents, at least the number specified by *checkatleast* will be passed to the matchspy. This means that you can force the matchspy to be shown all matching documents by setting *checkatleast* to the number of documents in the database.

**Parameters:**

*spy* The [MatchSpy](#) subclass to add. The caller must ensure that this remains valid while the [Enquire](#) object remains active, or until [clear\\_matchspies\(\)](#) is called.

**7.17.3.2 ESet Xapian::Enquire::get\_eset (Xapian::termcount *maxitems*, const RSet & *omrset*, int *flags*, double *k*, const Xapian::ExpandDecider \* *edecider*, Xapian::weight *min\_wt*) const**

Get the expand set for the given rset.

**Parameters:**

*maxitems* the maximum number of items to return.

*omrset* the relevance set to use when performing the expand operation.

*flags* zero or more of these values |'ed together:

- [Xapian::Enquire::INCLUDE\\_QUERY\\_TERMS](#) query terms may be returned from expand
- [Xapian::Enquire::USE\\_EXACT\\_TERM\\_FREQ](#) for multi dbs, calculate the exact termfreq; otherwise an approximation is used which can greatly improve efficiency, but still returns good results.

*k* the parameter *k* in the query expansion algorithm (default is 1.0)  
*edecider* a decision functor to use to decide whether a given term should be put in the [ESet](#)  
*min\_wt* the minimum weight for included terms

**Returns:**

An [ESet](#) object containing the results of the expand.

**Exceptions:**

[Xapian::InvalidArgumentError](#) See class documentation.

### 7.17.3.3 ESet Xapian::Enquire::get\_eset (Xapian::termcount *maxitems*, const RSet & *omrset*, const Xapian::ExpandDecider \* *edecider*) const [inline]

Get the expand set for the given rset.

**Parameters:**

*maxitems* the maximum number of items to return.  
*omrset* the relevance set to use when performing the expand operation.  
*edecider* a decision functor to use to decide whether a given term should be put in the [ESet](#)

**Returns:**

An [ESet](#) object containing the results of the expand.

**Exceptions:**

[Xapian::InvalidArgumentError](#) See class documentation.

### 7.17.3.4 ESet Xapian::Enquire::get\_eset (Xapian::termcount *maxitems*, const RSet & *omrset*, int *flags* = 0, double *k* = 1.0, const Xapian::ExpandDecider \* *edecider* = 0) const

Get the expand set for the given rset.

**Parameters:**

*maxitems* the maximum number of items to return.  
*omrset* the relevance set to use when performing the expand operation.  
*flags* zero or more of these values | -ed together:

- Xapian::Enquire::INCLUDE\_QUERY\_TERMS query terms may be returned from expand

- `Xapian::Enquire::USE_EXACT_TERMFREQ` for multi dbs, calculate the exact termfreq; otherwise an approximation is used which can greatly improve efficiency, but still returns good results.

*k* the parameter k in the query expansion algorithm (default is 1.0)

*edecider* a decision functor to use to decide whether a given term should be put in the [ESet](#)

*min\_wt* the minimum weight for included terms

#### Returns:

An [ESet](#) object containing the results of the expand.

#### Exceptions:

[Xapian::InvalidArgumentError](#) See class documentation.

#### 7.17.3.5 TermIterator Xapian::Enquire::get\_matching\_terms\_begin (const MSetIterator & it) const

Get terms which match a given document, by match set item.

This method returns the terms in the current query which match the given document.

If the underlying database has suitable support, using this call (rather than passing a [Xapian::docid](#)) will enable the system to ensure that the correct data is returned, and that the document has not been deleted or changed since the query was performed.

#### Parameters:

*it* The iterator for which to retrieve the matching terms.

#### Returns:

An iterator returning the terms which match the document. The terms will be returned (as far as this makes any sense) in the same order as the terms in the query. Terms will not occur more than once, even if they do in the query.

#### Exceptions:

[Xapian::InvalidArgumentError](#) See class documentation.

[Xapian::DocNotFoundError](#) The document specified could not be found in the database.

#### 7.17.3.6 TermIterator Xapian::Enquire::get\_matching\_terms\_begin (Xapian::docid did) const

Get terms which match a given document, by document id.

This method returns the terms in the current query which match the given document.

It is possible for the document to have been removed from the database between the time it is returned in an [MSet](#), and the time that this call is made. If possible, you should specify an [MSetIterator](#) instead of a [Xapian::docid](#), since this will enable database backends with suitable support to prevent this occurring.

Note that a query does not need to have been run in order to make this call.

#### Parameters:

*did* The document id for which to retrieve the matching terms.

#### Returns:

An iterator returning the terms which match the document. The terms will be returned (as far as this makes any sense) in the same order as the terms in the query. Terms will not occur more than once, even if they do in the query.

#### Exceptions:

[Xapian::InvalidArgumentError](#) See class documentation.

[Xapian::DocNotFoundError](#) The document specified could not be found in the database.

**7.17.3.7** `MSet Xapian::Enquire::get_mset (Xapian::doccount first,  
Xapian::doccount maxitems, const RSet * omrset, const MatchDecider  
* mdecider = 0) const` [inline]

Get (a portion of) the match set for the current query.

#### Parameters:

*first* the first item in the result set to return. A value of zero corresponds to the first item returned being that with the highest score. A value of 10 corresponds to the first 10 items being ignored, and the returned items starting at the eleventh.

*maxitems* the maximum number of items to return. If you want all matches, then you can pass the result of calling `get_doccount()` on the [Database](#) object (though if you are doing this so you can filter results, you are likely to get much better performance by using Xapian's match-time filtering features instead). You can pass 0 for *maxitems* which will give you an empty [MSet](#) with valid statistics (such as `get_matches_estimated()`) calculated without looking at any postings, which is very quick, but means the estimates may be more approximate and the bounds may be much looser.

*checkatleast* the minimum number of items to check. Because the matcher optimises, it won't consider every document which might match, so the total number of matches is estimated. Setting *checkatleast* forces it to consider at least this many matches and so allows for reliable paging links.

*omrset* the relevance set to use when performing the query.

*mdecider* a decision functor to use to decide whether a given document should be put in the [MSet](#).

*matchspy* a decision functor to use to decide whether a given document should be put in the [MSet](#). The matchspy is applied to every document which is a potential candidate for the [MSet](#), so if there are checkatleast or more such documents, the matchspy will see at least checkatleast. The mdecider is assumed to be a relatively expensive test so may be applied in a lazier fashion.

### Deprecated

this parameter is deprecated - use the newer [MatchSpy](#) class and [add\\_matchspy\(\)](#) method instead.

### Returns:

A [Xapian::MSet](#) object containing the results of the query.

### Exceptions:

[Xapian::InvalidArgumentError](#) See class documentation.

#### 7.17.3.8 MSet Xapian::Enquire::get\_mset (Xapian::doccount *first*, Xapian::doccount *maxitems*, Xapian::doccount *checkatleast*, const RSet \* *omrset*, const MatchDecider \* *mdecider*, const MatchDecider \* *matchspy*) const

Get (a portion of) the match set for the current query.

### Parameters:

*first* the first item in the result set to return. A value of zero corresponds to the first item returned being that with the highest score. A value of 10 corresponds to the first 10 items being ignored, and the returned items starting at the eleventh.

*maxitems* the maximum number of items to return. If you want all matches, then you can pass the result of calling [get\\_doccount\(\)](#) on the [Database](#) object (though if you are doing this so you can filter results, you are likely to get much better performance by using Xapian's match-time filtering features instead). You can pass 0 for maxitems which will give you an empty [MSet](#) with valid statistics (such as [get\\_matches\\_estimated\(\)](#)) calculated without looking at any postings, which is very quick, but means the estimates may be more approximate and the bounds may be much looser.

*checkatleast* the minimum number of items to check. Because the matcher optimises, it won't consider every document which might match, so the total number of matches is estimated. Setting checkatleast forces it to consider at least this many matches and so allows for reliable paging links.

*omrset* the relevance set to use when performing the query.

*mdecider* a decision functor to use to decide whether a given document should be put in the [MSet](#).

*matchspy* a decision functor to use to decide whether a given document should be put in the [MSet](#). The matchspy is applied to every document which is a potential candidate for the [MSet](#), so if there are atleast or more such documents, the matchspy will see at least atleast. The mdecider is assumed to be a relatively expensive test so may be applied in a lazier fashion.

### Deprecated

this parameter is deprecated - use the newer [MatchSpy](#) class and [add\\_matchspy\(\)](#) method instead.

### Returns:

A [Xapian::MSet](#) object containing the results of the query.

### Exceptions:

[Xapian::InvalidArgumentError](#) See class documentation.

#### 7.17.3.9 MSet Xapian::Enquire::get\_mset (Xapian::doccount *first*, Xapian::doccount *maxitems*, Xapian::doccount *checkatleast* = 0, const RSet \* *omrset* = 0, const MatchDecider \* *mdecider* = 0) const

Get (a portion of) the match set for the current query.

### Parameters:

*first* the first item in the result set to return. A value of zero corresponds to the first item returned being that with the highest score. A value of 10 corresponds to the first 10 items being ignored, and the returned items starting at the eleventh.

*maxitems* the maximum number of items to return. If you want all matches, then you can pass the result of calling [get\\_doccount\(\)](#) on the [Database](#) object (though if you are doing this so you can filter results, you are likely to get much better performance by using Xapian's match-time filtering features instead). You can pass 0 for maxitems which will give you an empty [MSet](#) with valid statistics (such as [get\\_matches\\_estimated\(\)](#)) calculated without looking at any postings, which is very quick, but means the estimates may be more approximate and the bounds may be much looser.

*checkatleast* the minimum number of items to check. Because the matcher optimises, it won't consider every document which might match, so the total number of matches is estimated. Setting checkatleast forces it to consider at least this many matches and so allows for reliable paging links.

*omrset* the relevance set to use when performing the query.

*mdecider* a decision functor to use to decide whether a given document should be put in the [MSet](#).

*matchspy* a decision functor to use to decide whether a given document should be put in the [MSet](#). The matchspy is applied to every document which is a

potential candidate for the [MSet](#), so if there are checkatleast or more such documents, the matchspy will see at least checkatleast. The mdecider is assumed to be a relatively expensive test so may be applied in a lazier fashion.

### Deprecated

this parameter is deprecated - use the newer [MatchSpy](#) class and [add\\_matchspy\(\)](#) method instead.

### Returns:

A [Xapian::MSet](#) object containing the results of the query.

### Exceptions:

[Xapian::InvalidArgumentError](#) See class documentation.

#### 7.17.3.10 const Xapian::Query& Xapian::Enquire::get\_query () const

Get the query which has been set.

This is only valid after [set\\_query\(\)](#) has been called.

### Exceptions:

[Xapian::InvalidArgumentError](#) will be thrown if query has not yet been set.

#### 7.17.3.11 void Xapian::Enquire::set\_collapse\_key (Xapian::valueno collapse\_key, Xapian::doccount collapse\_max = 1)

Set the collapse key to use for queries.

### Parameters:

*collapse\_key* value number to collapse on - at most one [MSet](#) entry with each particular value will be returned (default is [Xapian::BAD\\_VALUENO](#) which means no collapsing).

*collapse\_max* Max number of items with the same key to leave after collapsing (default 1).

The [MSet](#) returned by [get\\_mset\(\)](#) will have only the "best" (at most) *collapse\_max* entries with each particular value of *collapse\_key* ("best" being highest ranked - i.e. highest weight or highest sorting key).

An example use might be to create a value for each document containing an MD5 hash of the document contents. Then duplicate documents from different sources can be eliminated at search time by collapsing with *collapse\_max* = 1 (it's better to eliminate duplicates at index time, but this may not be always be possible - for example the search may be over more than one [Xapian](#) database).

Another use is to group matches in a particular category (e.g. you might collapse a mailing list search on the Subject: so that there's only one result per discussion thread). In this case you can use `get_collapse_count()` to give the user some idea how many other results there are. And if you index the Subject: as a boolean term as well as putting it in a value, you can offer a link to a non-collapsed search restricted to that thread using a boolean filter.

#### 7.17.3.12 `void Xapian::Enquire::set_cutoff (Xapian::percent percent_cutoff, Xapian::weight weight_cutoff = 0)`

Set the percentage and/or weight cutoffs.

##### Parameters:

*percent\_cutoff* Minimum percentage score for returned documents. If a document has a lower percentage score than this, it will not appear in the [MSet](#). If your intention is to return only matches which contain all the terms in the query, then it's more efficient to use [Xapian::Query::OP\\_AND](#) instead of [Xapian::Query::OP\\_OR](#) in the query than to use `set_cutoff(100)`. (default 0 => no percentage cut-off).

*weight\_cutoff* Minimum weight for a document to be returned. If a document has a lower score than this, it will not appear in the [MSet](#). It is usually only possible to choose an appropriate weight for cutoff based on the results of a previous run of the same query; this is thus mainly useful for alerting operations. The other potential use is with a user specified weighting scheme. (default 0 => no weight cut-off).

#### 7.17.3.13 `void Xapian::Enquire::set_docid_order (docid_order order)`

Set the direction in which documents are ordered by document id in the returned [MSet](#).

This order only has an effect on documents which would otherwise have equal rank. For a weighted probabilistic match with no sort value, this means documents with equal weight. For a boolean match, with no sort value, this means all documents. And if a sort value is used, this means documents with equal sort value (and also equal weight if ordering on relevance after the sort).

##### Parameters:

*order* This can be:

- `Xapian::Enquire::ASCENDING` docids sort in ascending order (default)
- `Xapian::Enquire::DESCENDING` docids sort in descending order
- `Xapian::Enquire::DONT_CARE` docids sort in whatever order is most efficient for the backend

Note: If you add documents in strict date order, then a boolean search - i.e. `set_weighting_scheme(Xapian::BoolWeight())` - with `set_docid_order(Xapian::Enquire::DESCENDING)` is a very efficient way to perform "sort by date, newest first".



**7.17.3.14 void Xapian::Enquire::set\_query (const Xapian::Query & *query*, Xapian::termcount *qlen* = 0)**

Set the query to run.

**Parameters:**

*query* the new query to run.

*qlen* the query length to use in weight calculations - by default the sum of the wqf of all terms is used.

**7.17.3.15 void Xapian::Enquire::set\_sort\_by\_key (Xapian::KeyMaker \* *sorter*, bool *reverse*)**

Set the sorting to be by key generated from values only.

**Parameters:**

*sorter* The functor to use for generating keys.

*reverse* If true, reverses the sort order.

**7.17.3.16 void Xapian::Enquire::set\_sort\_by\_key\_then\_relevance (Xapian::KeyMaker \* *sorter*, bool *reverse*)**

Set the sorting to be by keys generated from values, then by relevance for documents with identical keys.

**Parameters:**

*sorter* The functor to use for generating keys.

*reverse* If true, reverses the sort order.

**7.17.3.17 void Xapian::Enquire::set\_sort\_by\_relevance ()**

Set the sorting to be by relevance only.

This is the default.

**7.17.3.18 void Xapian::Enquire::set\_sort\_by\_relevance\_then\_key (Xapian::KeyMaker \* *sorter*, bool *reverse*)**

Set the sorting to be by relevance, then by keys generated from values.

Note that with the default BM25 weighting scheme parameters, non-identical documents will rarely have the same weight, so this setting will give very similar results to [set\\_sort\\_by\\_relevance\(\)](#). It becomes more useful with particular BM25 parameter settings (e.g. BM25Weight(1,0,1,0,0)) or custom weighting schemes.

**Parameters:**

*sorter* The functor to use for generating keys.

*reverse* If true, reverses the sort order.

**7.17.3.19 void Xapian::Enquire::set\_sort\_by\_relevance\_then\_value (Xapian::valueno *sort\_key*, bool *reverse*)**

Set the sorting to be by relevance then value.

Note that sorting by values uses a string comparison, so to use this to sort by a numeric value you'll need to store the numeric values in a manner which sorts appropriately. For example, you could use [Xapian::sortable\\_serialise\(\)](#) (which works for floating point numbers as well as integers), or store numbers padded with leading zeros or spaces, or with the number of digits prepended.

Note that with the default BM25 weighting scheme parameters, non-identical documents will rarely have the same weight, so this setting will give very similar results to [set\\_sort\\_by\\_relevance\(\)](#). It becomes more useful with particular BM25 parameter settings (e.g. `BM25Weight(1,0,1,0,0)`) or custom weighting schemes.

**Parameters:**

*sort\_key* value number to sort on.

*reverse* If true, reverses the sort order.

**7.17.3.20 void Xapian::Enquire::set\_sort\_by\_value (Xapian::valueno *sort\_key*, bool *reverse*)**

Set the sorting to be by value only.

Note that sorting by values uses a string comparison, so to use this to sort by a numeric value you'll need to store the numeric values in a manner which sorts appropriately. For example, you could use [Xapian::sortable\\_serialise\(\)](#) (which works for floating point numbers as well as integers), or store numbers padded with leading zeros or spaces, or with the number of digits prepended.

**Parameters:**

*sort\_key* value number to sort on.

*reverse* If true, reverses the sort order.

**7.17.3.21 void Xapian::Enquire::set\_sort\_by\_value\_then\_relevance (Xapian::valueno *sort\_key*, bool *reverse*)**

Set the sorting to be by value, then by relevance for documents with the same value.

Note that sorting by values uses a string comparison, so to use this to sort by a numeric value you'll need to store the numeric values in a manner which sorts appropriately. For

example, you could use [Xapian::sortable\\_serialise\(\)](#) (which works for floating point numbers as well as integers), or store numbers padded with leading zeros or spaces, or with the number of digits prepended.

**Parameters:**

*sort\_key* value number to sort on.

*reverse* If true, reverses the sort order.

**7.17.3.22 void Xapian::Enquire::set\_weighting\_scheme (const Weight & weight\_)**

Set the weighting scheme to use for queries.

**Parameters:**

*weight\_* the new weighting scheme. If no weighting scheme is specified, the default is BM25 with the default parameters.

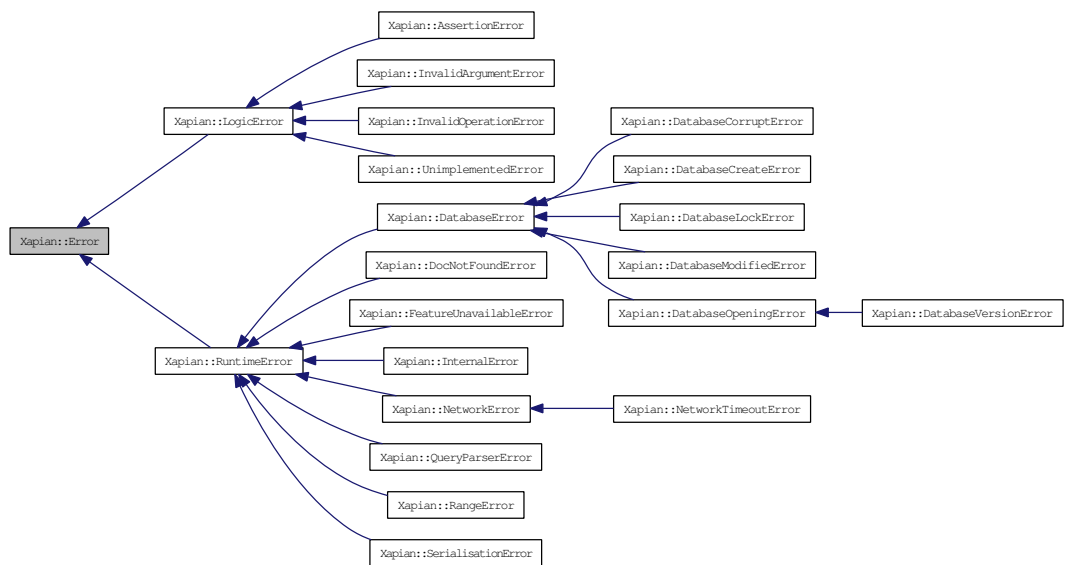
The documentation for this class was generated from the following file:

- [xapian/enquire.h](#)

## 7.18 Xapian::Error Class Reference

All exceptions thrown by [Xapian](#) are subclasses of [Xapian::Error](#).

Inheritance diagram for [Xapian::Error](#):



### Public Member Functions

- `const char * get\_type () const`  
The type of this error (e.g. "DocNotFoundError").
- `const std::string & get\_msg () const`  
Message giving details of the error, intended for human consumption.
- `const std::string & get\_context () const`  
Optional context information.
- `const char * get\_error\_string () const`  
Returns any system error string associated with this exception.
- `std::string get\_description () const`  
Return a string describing this object.

### 7.18.1 Detailed Description

All exceptions thrown by [Xapian](#) are subclasses of [Xapian::Error](#).

This class can not be instantiated directly - instead a subclass should be used.

## 7.18.2 Member Function Documentation

### 7.18.2.1 `const std::string& Xapian::Error::get_context () const` [inline]

Optional context information.

This context is intended for use by [Xapian::ErrorHandler](#) (for example so it can know which remote server is unreliable and report the problem and remove that server from those being searched). But it's typically a plain-text string, and so also fit for human consumption.

### 7.18.2.2 `const char* Xapian::Error::get_error_string () const`

Returns any system error string associated with this exception.

The system error string may come from `errno`, `h_errno` (on UNIX), or `GetLastError()` (on MS Windows). If there is no associated system error string, `NULL` is returned.

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

- [xapian/error.h](#)

## 7.19 Xapian::ErrorHandler Class Reference

Decide if a [Xapian::Error](#) exception should be ignored.

### Public Member Functions

- [ErrorHandler](#) ()  
*Default constructor.*
- virtual [~ErrorHandler](#) ()  
*We require a virtual destructor because we have virtual methods.*
- void [operator\(\)](#) ([Xapian::Error](#) &error)  
*Handle a [Xapian::Error](#) object.*

#### 7.19.1 Detailed Description

Decide if a [Xapian::Error](#) exception should be ignored.

You can create your own subclass of this class and pass in an instance of it when you construct a [Xapian::Enquire](#) object. [Xapian::Error](#) exceptions which happen during the match process are passed to this object and it can decide whether they should propagate or whether [Enquire](#) should attempt to continue.

The motivation is to allow searching over remote databases to handle a remote server which has died (both to allow results to be returned, and also so that such errors can be logged and dead servers temporarily removed from use).

#### 7.19.2 Member Function Documentation

##### 7.19.2.1 void Xapian::ErrorHandler::operator() (Xapian::Error & error)

Handle a [Xapian::Error](#) object.

This method is called when a [Xapian::Error](#) object is thrown and caught inside [Enquire](#). If this is the first [ErrorHandler](#) that the [Error](#) has been passed to, then the [handle\\_error\(\)](#) virtual method is called, which allows the API user to decide how to handle the error.

##### Parameters:

**error** The [Xapian::Error](#) object under consideration.

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

- [xapian/errorhandler.h](#)

## 7.20 Xapian::ESet Class Reference

Class representing an ordered set of expand terms (an [ESet](#)).

### Public Member Functions

- [ESet](#) ()  
*Construct an empty [ESet](#).*
- [~ESet](#) ()  
*Destructor.*
- [ESet](#) (const [ESet](#) &other)  
*Copying is allowed (and is cheap).*
- void [operator=](#) (const [ESet](#) &other)  
*Assignment is allowed (and is cheap).*
- [Xapian::termcount](#) [get\\_ebound](#) () const  
*A lower bound on the number of terms which are in the full set of results of the expand.*
- [Xapian::termcount](#) [size](#) () const  
*The number of terms in this E-Set.*
- [Xapian::termcount](#) [max\\_size](#) () const  
*Required to allow use as an STL container.*
- bool [empty](#) () const  
*Test if this E-Set is empty.*
- void [swap](#) ([ESet](#) &other)  
*Swap the E-Set we point to with another.*
- [ESetIterator](#) [begin](#) () const  
*Iterator for the terms in this E-Set.*
- [ESetIterator](#) [end](#) () const  
*End iterator corresponding to [begin\(\)](#).*
- [ESetIterator](#) [back](#) () const  
*Iterator pointing to the last element of this E-Set.*
- [ESetIterator](#) [operator\[ \]](#) ([Xapian::termcount](#) i) const  
*This returns the term at position i in this E-Set.*
- std::string [get\\_description](#) () const

*Return a string describing this object.*

### 7.20.1 Detailed Description

Class representing an ordered set of expand terms (an [ESet](#)).

This set represents the results of an expand operation, which is performed by [Xapian::Enquire::get\\_eset\(\)](#).

### 7.20.2 Member Function Documentation

#### 7.20.2.1 `Xapian::termcount Xapian::ESet::get_ebound () const`

A lower bound on the number of terms which are in the full set of results of the expand.

This will be greater than or equal to [size\(\)](#)

#### 7.20.2.2 `Xapian::termcount Xapian::ESet::max_size () const` `[inline]`

Required to allow use as an STL container.

#### 7.20.2.3 `ESetIterator Xapian::ESet::operator[] (Xapian::termcount i) const`

This returns the term at position *i* in this E-Set.

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

- [xapian/enquire.h](#)



## 7.21 Xapian::ESetIterator Class Reference

Iterate through terms in the [ESet](#).

### Public Types

- typedef std::bidirectional\_iterator\_tag [iterator\\_category](#)  
*Allow use as an STL iterator.*
- typedef std::string [value\\_type](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::termcount\\_diff](#) [difference\\_type](#)  
*Allow use as an STL iterator.*
- typedef std::string \* [pointer](#)  
*Allow use as an STL iterator.*
- typedef std::string & [reference](#)  
*Allow use as an STL iterator.*

### Public Member Functions

- [ESetIterator](#) ()  
*Create an uninitialised iterator; this cannot be used, but is convenient syntactically.*
- [ESetIterator](#) (const [ESetIterator](#) &other)  
*Copying is allowed (and is cheap).*
- void [operator=](#) (const [ESetIterator](#) &other)  
*Assignment is allowed (and is cheap).*
- [ESetIterator](#) & [operator++](#) ()  
*Advance the iterator.*
- [ESetIterator](#) [operator++](#) (int)  
*Advance the iterator (postfix variant).*
- [ESetIterator](#) & [operator--](#) ()  
*Decrement the iterator.*
- [ESetIterator](#) [operator--](#) (int)  
*Decrement the iterator (postfix variant).*
- const std::string & [operator\\*](#) () const

*Get the term for the current position.*

- [Xapian::weight](#) [get\\_weight](#) () const  
*Get the weight of the term at the current position.*
- `std::string` [get\\_description](#) () const  
*Return a string describing this object.*

## Friends

- `bool` [operator==](#) (const [ESetIterator](#) &a, const [ESetIterator](#) &b)  
*Equality test for [ESetIterator](#) objects.*
- `bool` [operator!=](#) (const [ESetIterator](#) &a, const [ESetIterator](#) &b)  
*Inequality test for [ESetIterator](#) objects.*

### 7.21.1 Detailed Description

Iterate through terms in the [ESet](#).

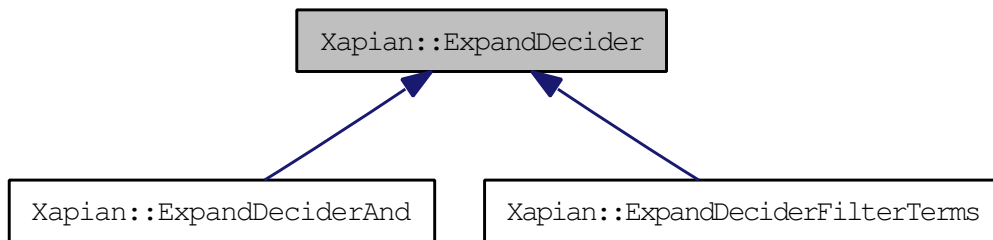
The documentation for this class was generated from the following file:

- [xapian/enquire.h](#)

## 7.22 Xapian::ExpandDecider Class Reference

Virtual base class for expand decider functor.

Inheritance diagram for Xapian::ExpandDecider:



### Public Member Functions

- virtual bool [operator\(\)](#) (const std::string &term) const =0  
*Do we want this term in the [ESet](#)?*
- virtual [~ExpandDecider](#) ()  
*Virtual destructor, because we have virtual methods.*

### 7.22.1 Detailed Description

Virtual base class for expand decider functor.

### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 virtual Xapian::ExpandDecider::~~ExpandDecider () [virtual]

Virtual destructor, because we have virtual methods.

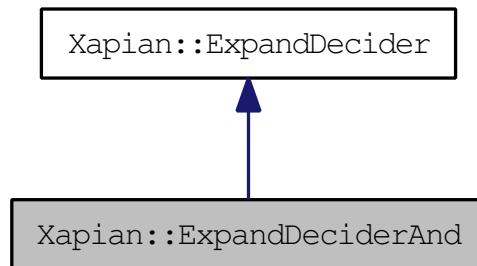
The documentation for this class was generated from the following file:

- [xapian/expanddecider.h](#)

## 7.23 Xapian::ExpandDeciderAnd Class Reference

[ExpandDecider](#) subclass which rejects terms using two [ExpandDeciders](#).

Inheritance diagram for Xapian::ExpandDeciderAnd:



### Public Member Functions

- [ExpandDeciderAnd](#) (const [ExpandDecider](#) &first\_, const [ExpandDecider](#) &second\_)  
*Terms will be checked with first, and if accepted, then checked with second.*
- [ExpandDeciderAnd](#) (const [ExpandDecider](#) \*first\_, const [ExpandDecider](#) \*second\_)  
*Compatibility method.*
- virtual bool [operator\(\)](#) (const std::string &term) const  
*Do we want this term in the [ESet](#)?*

#### 7.23.1 Detailed Description

[ExpandDecider](#) subclass which rejects terms using two [ExpandDeciders](#).

Terms are only accepted if they are accepted by both of the specified [ExpandDecider](#) objects.

#### 7.23.2 Constructor & Destructor Documentation

**7.23.2.1** [Xapian::ExpandDeciderAnd::ExpandDeciderAnd](#) (const [ExpandDecider](#) \*first\_, const [ExpandDecider](#) \*second\_) [inline]

Compatibility method.

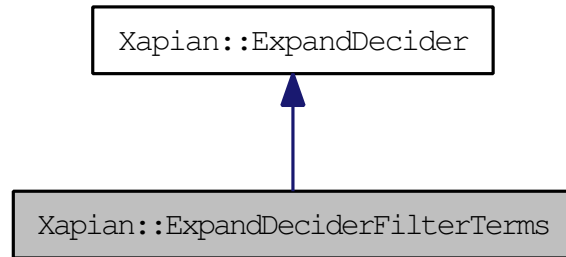
The documentation for this class was generated from the following file:

- xapian/[expanddecider.h](#)

## 7.24 Xapian::ExpandDeciderFilterTerms Class Reference

[ExpandDecider](#) subclass which rejects terms in a specified list.

Inheritance diagram for Xapian::ExpandDeciderFilterTerms:



### Public Member Functions

- `template<class Iterator >`  
[ExpandDeciderFilterTerms](#) (Iterator reject\_begin, Iterator reject\_end)  
*The two iterators specify a list of terms to be rejected.*
- `virtual bool operator() (const std::string &term) const`  
*Do we want this term in the [ESet](#)?*

#### 7.24.1 Detailed Description

[ExpandDecider](#) subclass which rejects terms in a specified list.

[ExpandDeciderFilterTerms](#) provides an easy way to filter out terms from a fixed list when generating an [ESet](#).

#### 7.24.2 Constructor & Destructor Documentation

- 7.24.2.1** `template<class Iterator >`  
**Xapian::ExpandDeciderFilterTerms::ExpandDeciderFilterTerms**  
 (Iterator reject\_begin, Iterator reject\_end) `[inline]`

The two iterators specify a list of terms to be rejected.

*reject\_begin* and *reject\_end* can be any `input_iterator` type which returns `std::string` or `char *` (e.g. [TermIterator](#) or `char **`).

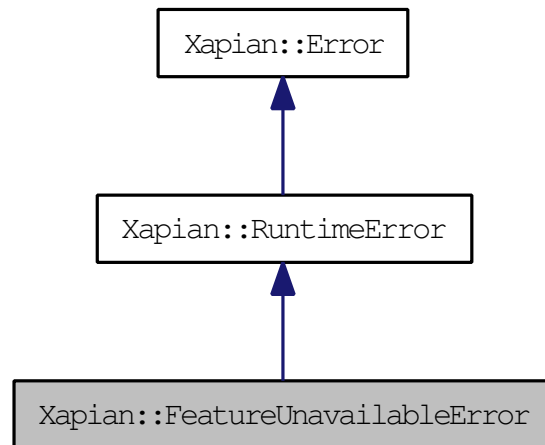
The documentation for this class was generated from the following file:

- [xapian/expanddecider.h](#)

## 7.25 Xapian::FeatureUnavailableError Class Reference

Indicates an attempt to use a feature which is unavailable.

Inheritance diagram for Xapian::FeatureUnavailableError:



### Public Member Functions

- [FeatureUnavailableError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)  
*General purpose constructor which allows setting errno.*
- [FeatureUnavailableError](#) (const std::string &msg\_, int errno\_)  
*Construct from message and errno value.*

### 7.25.1 Detailed Description

Indicates an attempt to use a feature which is unavailable.

Typically a feature is unavailable because it wasn't compiled in, or because it requires other software or facilities which aren't available.

### 7.25.2 Constructor & Destructor Documentation

#### 7.25.2.1 Xapian::FeatureUnavailableError::FeatureUnavailableError (const std::string &msg\_, const std::string &context\_ = std::string(), int *errno*\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

#### 7.25.2.2 Xapian::FeatureUnavailableError::FeatureUnavailableError (const std::string & *msg\_*, int *errno\_*) [inline]

Construct from message and errno value.

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

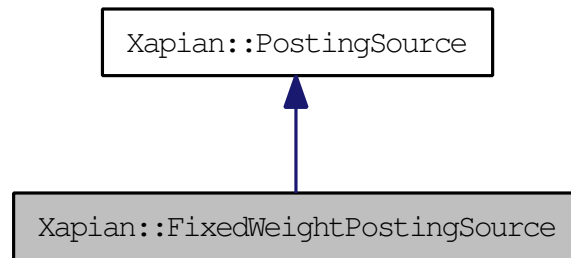
- [xapian/error.h](#)



## 7.26 Xapian::FixedWeightPostingSource Class Reference

A posting source which returns a fixed weight for all documents.

Inheritance diagram for Xapian::FixedWeightPostingSource:



### Public Member Functions

- [FixedWeightPostingSource](#) ([Xapian::weight](#) wt)  
*Construct a [FixedWeightPostingSource](#).*
- [Xapian::doccount](#) [get\\_termfreq\\_min](#) () const  
*A lower bound on the number of documents this object can return.*
- [Xapian::doccount](#) [get\\_termfreq\\_est](#) () const  
*An estimate of the number of documents this object can return.*
- [Xapian::doccount](#) [get\\_termfreq\\_max](#) () const  
*An upper bound on the number of documents this object can return.*
- [Xapian::weight](#) [get\\_weight](#) () const  
*Return the weight contribution for the current document.*
- void [next](#) ([Xapian::weight](#) min\_wt)  
*Advance the current position to the next matching document.*
- void [skip\\_to](#) ([Xapian::docid](#) min\_docid, [Xapian::weight](#) min\_wt)  
*Advance to the specified docid.*
- bool [check](#) ([Xapian::docid](#) min\_docid, [Xapian::weight](#) min\_wt)  
*Check if the specified docid occurs.*
- bool [at\\_end](#) () const  
*Return true if the current position is past the last entry in this list.*

- [Xapian::docid](#) [get\\_docid](#) () const  
*Return the current docid.*
- [FixedWeightPostingSource](#) \* [clone](#) () const  
*Clone the posting source.*
- std::string [name](#) () const  
*Name of the posting source class.*
- std::string [serialise](#) () const  
*Serialise object parameters into a string.*
- [FixedWeightPostingSource](#) \* [unserialise](#) (const std::string &s) const  
*Create object given string serialisation returned by [serialise\(\)](#).*
- void [init](#) (const [Database](#) &db\_)  
*Set this [PostingSource](#) to the start of the list of postings.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.26.1 Detailed Description

A posting source which returns a fixed weight for all documents.

This returns entries for all documents in the given database, with a fixed weight (specified by a parameter to the constructor).

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 [Xapian::FixedWeightPostingSource::FixedWeightPostingSource](#) ([Xapian::weight](#) *wt*)

Construct a [FixedWeightPostingSource](#).

##### Parameters:

*wt* The fixed weight to return.

### 7.26.3 Member Function Documentation

#### 7.26.3.1 [bool](#) [Xapian::FixedWeightPostingSource::at\\_end](#) () const [virtual]

Return true if the current position is past the last entry in this list.

At least one of [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#) will be called before this method is first called.

Implements [Xapian::PostingSource](#).

### 7.26.3.2 `bool Xapian::FixedWeightPostingSource::check (Xapian::docid did, Xapian::weight min_wt)` [virtual]

Check if the specified docid occurs.

The caller is required to ensure that the specified document id *did* actually exists in the database. If it does, it must move to that document id, and return true. If it does not, it may either:

- return true, having moved to a definite position (including "at\_end"), which must be the same position as [skip\\_to\(\)](#) would have moved to.

or

- return false, having moved to an "indeterminate" position, such that a subsequent call to [next\(\)](#) or [skip\\_to\(\)](#) will move to the next matching position after *did*.

Generally, this method should act like [skip\\_to\(\)](#) and return true if that can be done at little extra cost.

Otherwise it should simply check if a particular docid is present, returning true if it is, and false if it isn't.

The default implementation calls [skip\\_to\(\)](#) and always returns true.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

Reimplemented from [Xapian::PostingSource](#).

### 7.26.3.3 `FixedWeightPostingSource*` `Xapian::FixedWeightPostingSource::clone () const` [virtual]

Clone the posting source.

The clone should inherit the configuration of the parent, but need not inherit the state. ie, the clone does not need to be in the same iteration position as the original: the matcher will always call [init\(\)](#) on the clone before attempting to move the iterator, or read the information about the current position of the iterator.

This may return NULL to indicate that cloning is not supported. In this case, the [PostingSource](#) may only be used with a single-database search.

The default implementation returns NULL.

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented from [Xapian::PostingSource](#).

#### 7.26.3.4 `std::string Xapian::FixedWeightPostingSource::get_description () const` [virtual]

Return a string describing this object.

This default implementation returns a generic answer. This default is provided to avoid forcing those deriving their own [PostingSource](#) subclass from having to implement this (they may not care what [get\\_description\(\)](#) gives for their subclass).

Reimplemented from [Xapian::PostingSource](#).

#### 7.26.3.5 `Xapian::docid Xapian::FixedWeightPostingSource::get_docid () const` [virtual]

Return the current docid.

This method may assume that it will only be called when there is a "current document". See [get\\_weight\(\)](#) for details.

Note: in the case of a multi-database search, the returned docid should be in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

Implements [Xapian::PostingSource](#).

#### 7.26.3.6 `Xapian::doccount Xapian::FixedWeightPostingSource::get_termfreq_est () const` [virtual]

An estimate of the number of documents this object can return.

It must always be true that:

`get_termfreq_min() <= get_termfreq_est() <= get_termfreq_max()`

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implements [Xapian::PostingSource](#).

#### 7.26.3.7 `Xapian::doccount Xapian::FixedWeightPostingSource::get_termfreq_max () const` [virtual]

An upper bound on the number of documents this object can return.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implements [Xapian::PostingSource](#).

### 7.26.3.8 Xapian::doccount Xapian::FixedWeightPostingSource::get\_termfreq\_min () const [virtual]

A lower bound on the number of documents this object can return.

Xapian will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implements [Xapian::PostingSource](#).

### 7.26.3.9 Xapian::weight Xapian::FixedWeightPostingSource::get\_weight () const [virtual]

Return the weight contribution for the current document.

This default implementation always returns 0, for convenience when implementing "weight-less" [PostingSource](#) subclasses.

This method may assume that it will only be called when there is a "current document". In detail: Xapian will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time. It will also only call this if the [PostingSource](#) reports that it is pointing to a valid document (ie, it will not call it before calling at least one of [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#), and will ensure that the [PostingSource](#) is not at the end by calling [at\\_end\(\)](#)).

Reimplemented from [Xapian::PostingSource](#).

### 7.26.3.10 void Xapian::FixedWeightPostingSource::init (const Database & db) [virtual]

Set this [PostingSource](#) to the start of the list of postings.

This is called automatically by the matcher prior to each query being processed.

If a [PostingSource](#) is used for multiple searches, [init\(\)](#) will therefore be called multiple times, and must handle this by using the database passed in the most recent call.

#### Parameters:

**db** The database which the [PostingSource](#) should iterate through.

Note: the database supplied to this method must not be modified: in particular, the [reopen\(\)](#) method should not be called on it.

Note: in the case of a multi-database search, a separate [PostingSource](#) will be used for each database (the separate PostingSources will be obtained using [clone\(\)](#)), and each [PostingSource](#) will be passed one of the sub-databases as the *db* parameter here. The *db* parameter will therefore always refer to a single database. All docids passed to, or returned from, the [PostingSource](#) refer to docids in that single database, rather than in the multi-database.

Implements [Xapian::PostingSource](#).

### 7.26.3.11 `std::string Xapian::FixedWeightPostingSource::name () const` [virtual]

Name of the posting source class.

This is used when serialising and unserialising posting sources; for example, for performing remote searches.

If the subclass is in a C++ namespace, the namespace should be included in the name, using "::" as a separator. For example, for a [PostingSource](#) subclass called "FooPostingSource" in the "Xapian" namespace the result of this call should be "Xapian::FooPostingSource".

This should only be implemented if [serialise\(\)](#) and [unserialise\(\)](#) are also implemented. The default implementation returns an empty string.

If this returns an empty string, [Xapian](#) will assume that [serialise\(\)](#) and [unserialise\(\)](#) are not implemented.

Reimplemented from [Xapian::PostingSource](#).

### 7.26.3.12 `void Xapian::FixedWeightPostingSource::next (Xapian::weight min_wt)` [virtual]

Advance the current position to the next matching document.

The [PostingSource](#) starts before the first entry in the list, so [next\(\)](#) must be called before any methods which need the context of the current position.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

#### Parameters:

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Implements [Xapian::PostingSource](#).

### 7.26.3.13 `std::string Xapian::FixedWeightPostingSource::serialise () const` [virtual]

Serialise object parameters into a string.

The serialised parameters should represent the configuration of the posting source, but need not (indeed, should not) represent the current iteration state.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::PostingSource](#).

### 7.26.3.14 void Xapian::FixedWeightPostingSource::skip\_to (Xapian::docid *did*, Xapian::weight *min\_wt*) [virtual]

Advance to the specified docid.

If the specified docid isn't in the list, position ourselves on the first document after it (or [at\\_end\(\)](#) if no greater docids are present).

If the current position is already the specified docid, this method will leave the position unmodified.

If the specified docid is earlier than the current position, the behaviour is unspecified. A sensible behaviour would be to leave the current position unmodified, but it is also reasonable to move to the specified docid.

The default implementation calls [next\(\)](#) repeatedly, which works but [skip\\_to\(\)](#) can often be implemented much more efficiently.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

#### Parameters:

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Reimplemented from [Xapian::PostingSource](#).

### 7.26.3.15 FixedWeightPostingSource\* Xapian::FixedWeightPostingSource::unserialise (const std::string & *s*) const [virtual]

Create object given string serialisation returned by [serialise\(\)](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

#### Parameters:

*s* A serialised instance of this [PostingSource](#) subclass.

Reimplemented from [Xapian::PostingSource](#).

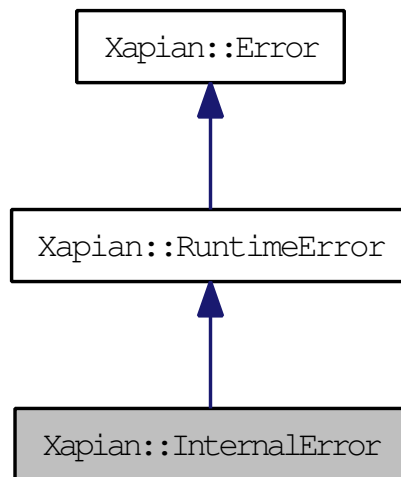
The documentation for this class was generated from the following file:

- [xapian/postingsource.h](#)

## 7.27 Xapian::InternalError Class Reference

[InternalError](#) indicates a runtime problem of some sort.

Inheritance diagram for Xapian::InternalError:



### Public Member Functions

- [InternalError](#) (const std::string &msg\_, const std::string &context\_=std::string(), int errno\_=0)

*General purpose constructor which allows setting errno.*

- [InternalError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.27.1 Detailed Description

[InternalError](#) indicates a runtime problem of some sort.

### 7.27.2 Constructor & Destructor Documentation

- #### 7.27.2.1 Xapian::InternalError::InternalError (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.



### 7.27.2.2 Xapian::InternalError::InternalError (const std::string & *msg\_*, int *errno\_*) [inline]

Construct from message and errno value.

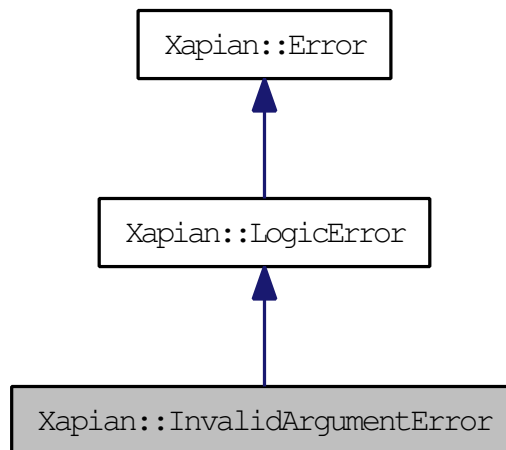
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.28 Xapian::InvalidArgumentError Class Reference

[InvalidArgumentError](#) indicates an invalid parameter value was passed to the API.

Inheritance diagram for Xapian::InvalidArgumentError:



### Public Member Functions

- [InvalidArgumentError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [InvalidArgumentError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.28.1 Detailed Description

[InvalidArgumentError](#) indicates an invalid parameter value was passed to the API.

### 7.28.2 Constructor & Destructor Documentation

- 7.28.2.1 Xapian::InvalidArgumentError::InvalidArgumentError (const std::string & msg\_, const std::string & context\_ = std::string(), int errno\_ = 0) [inline, explicit]**

General purpose constructor which allows setting errno.

### 7.28.2.2 Xapian::InvalidArgumentError::InvalidArgumentError (const std::string & msg\_, int errno\_) [inline]

Construct from message and errno value.

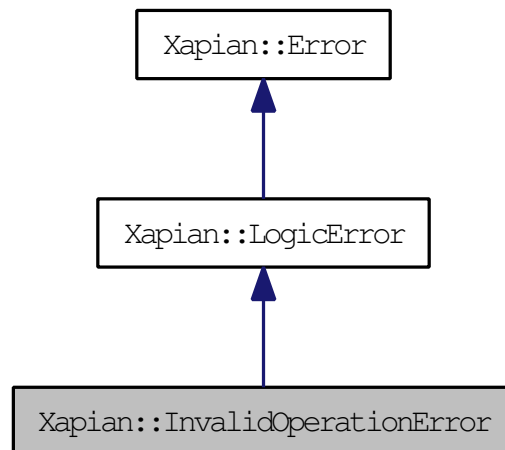
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.29 Xapian::InvalidOperationError Class Reference

[InvalidOperationError](#) indicates the API was used in an invalid way.

Inheritance diagram for Xapian::InvalidOperationError:



### Public Member Functions

- [InvalidOperationError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [InvalidOperationError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.29.1 Detailed Description

[InvalidOperationError](#) indicates the API was used in an invalid way.

### 7.29.2 Constructor & Destructor Documentation

- #### 7.29.2.1 Xapian::InvalidOperationError::InvalidOperationError (const std::string & msg\_, const std::string & context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

### 7.29.2.2 Xapian::InvalidOperationError::InvalidOperationError (const std::string & msg\_, int errno\_) [inline]

Construct from message and errno value.

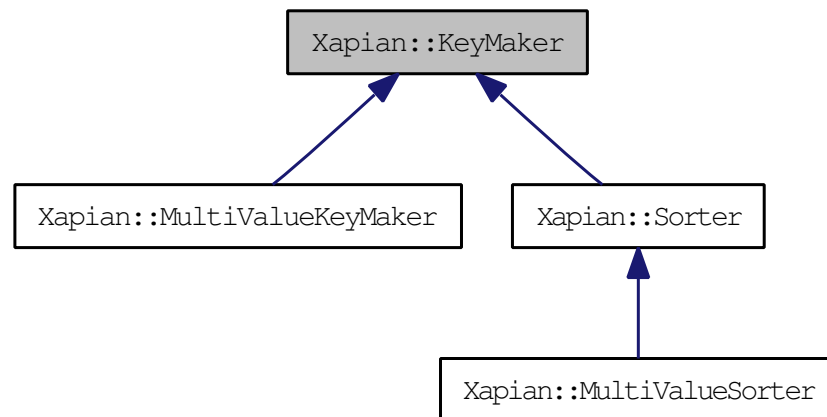
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.30 Xapian::KeyMaker Class Reference

Virtual base class for key making functors.

Inheritance diagram for Xapian::KeyMaker:



### Public Member Functions

- virtual std::string [operator\(\)](#) (const [Xapian::Document](#) &doc) const =0  
*This method takes a [Document](#) object and builds a key string from it.*
- virtual [~KeyMaker](#) ()  
*Virtual destructor, because we have virtual methods.*

#### 7.30.1 Detailed Description

Virtual base class for key making functors.

#### 7.30.2 Constructor & Destructor Documentation

##### 7.30.2.1 virtual Xapian::KeyMaker::~~KeyMaker () [virtual]

Virtual destructor, because we have virtual methods.

#### 7.30.3 Member Function Documentation

##### 7.30.3.1 virtual std::string Xapian::KeyMaker::operator() (const Xapian::Document & doc) const [pure virtual]

This method takes a [Document](#) object and builds a key string from it.

These keys are then used for ordering or collapsing matching documents.

Implemented in [Xapian::MultiValueKeyMaker](#), and [Xapian::MultiValueSorter](#).

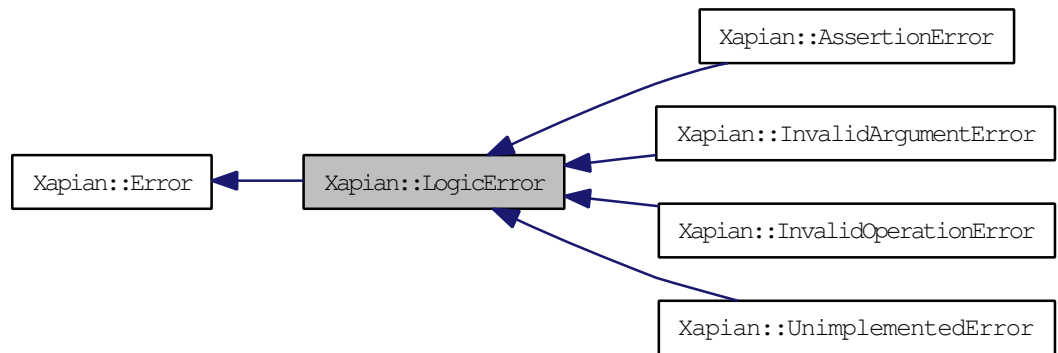
The documentation for this class was generated from the following file:

- [xapian/keymaker.h](#)

## 7.31 Xapian::LogicError Class Reference

The base class for exceptions indicating errors in the program logic.

Inheritance diagram for Xapian::LogicError:



### 7.31.1 Detailed Description

The base class for exceptions indicating errors in the program logic.

A subclass of [LogicError](#) will be thrown if [Xapian](#) detects a violation of a class invariant or a logical precondition or postcondition, etc.

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

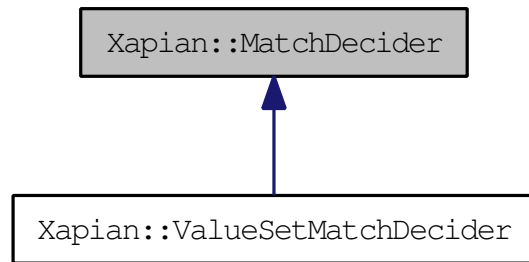
- [xapian/error.h](#)



## 7.32 Xapian::MatchDecider Class Reference

Base class for matcher decision functor.

Inheritance diagram for Xapian::MatchDecider:



### Public Member Functions

- virtual bool [operator\(\)](#) (const [Xapian::Document](#) &doc) const =0  
*Decide whether we want this document to be in the [MSet](#).*
- virtual [~MatchDecider](#) ()  
*Destructor.*

### 7.32.1 Detailed Description

Base class for matcher decision functor.

### 7.32.2 Member Function Documentation

#### 7.32.2.1 virtual bool Xapian::MatchDecider::operator() (const Xapian::Document & doc) const [pure virtual]

Decide whether we want this document to be in the [MSet](#).

Return true if the document is acceptable, or false if the document should be excluded from the [MSet](#).

Implemented in [Xapian::ValueSetMatchDecider](#).

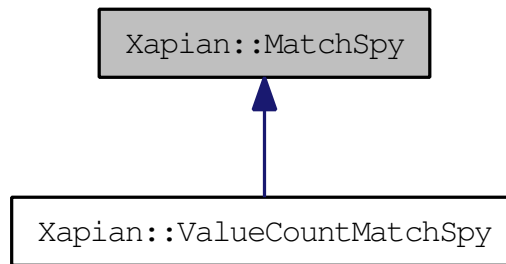
The documentation for this class was generated from the following file:

- [xapian/enquire.h](#)

## 7.33 Xapian::MatchSpy Class Reference

Abstract base class for match spies.

Inheritance diagram for Xapian::MatchSpy:



### Public Member Functions

- virtual [~MatchSpy](#) ()  
*Virtual destructor, because we have virtual methods.*
- virtual void [operator\(\)](#) (const [Xapian::Document](#) &doc, [Xapian::weight](#) wt)=0  
*Register a document with the match spy.*
- virtual [MatchSpy](#) \* [clone](#) () const  
*Clone the match spy.*
- virtual std::string [name](#) () const  
*Return the name of this match spy.*
- virtual std::string [serialise](#) () const  
*Return this object's parameters serialised as a single string.*
- virtual [MatchSpy](#) \* [unserialise](#) (const std::string &s, const [Registry](#) &context) const  
*Unserialise parameters.*
- virtual std::string [serialise\\_results](#) () const  
*Serialise the results of this match spy.*
- virtual void [merge\\_results](#) (const std::string &s)  
*Unserialise some results, and merge them into this matchspy.*
- virtual std::string [get\\_description](#) () const  
*Return a string describing this object.*

## Protected Member Functions

- [MatchSpy \(\)](#)

*Default constructor, needed by subclass constructors.*

### 7.33.1 Detailed Description

Abstract base class for match spies.

The subclasses will generally accumulate information seen during the match, to calculate aggregate functions, or other profiles of the matching documents.

### 7.33.2 Constructor & Destructor Documentation

#### 7.33.2.1 `virtual Xapian::MatchSpy::~~MatchSpy ()` [virtual]

Virtual destructor, because we have virtual methods.

### 7.33.3 Member Function Documentation

#### 7.33.3.1 `virtual MatchSpy* Xapian::MatchSpy::clone () const` [virtual]

Clone the match spy.

The clone should inherit the configuration of the parent, but need not inherit the state. ie, the clone does not need to be passed information about the results seen by the parent.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented in [Xapian::ValueCountMatchSpy](#).

#### 7.33.3.2 `virtual std::string Xapian::MatchSpy::get_description () const` [virtual]

Return a string describing this object.

This default implementation returns a generic answer, to avoid forcing those deriving their own [MatchSpy](#) subclasses from having to implement this (they may not care what [get\\_description\(\)](#) gives for their subclass).

Reimplemented in [Xapian::ValueCountMatchSpy](#).

### 7.33.3.3 **virtual void Xapian::MatchSpy::merge\_results (const std::string & s)** [virtual]

Unserialise some results, and merge them into this matchspy.

The order in which results are merged should not be significant, since this order is not specified (and will vary depending on the speed of the search in each sub-database).

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented in [Xapian::ValueCountMatchSpy](#).

### 7.33.3.4 **virtual std::string Xapian::MatchSpy::name () const** [virtual]

Return the name of this match spy.

This name is used by the remote backend. It is passed with the serialised parameters to the remote server so that it knows which class to create.

Return the full namespace-qualified name of your class here - if your class is called `MyApp::FooMatchSpy`, return `"MyApp::FooMatchSpy"` from this method.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented in [Xapian::ValueCountMatchSpy](#).

### 7.33.3.5 **virtual void Xapian::MatchSpy::operator() (const Xapian::Document & doc, Xapian::weight wt)** [pure virtual]

Register a document with the match spy.

This is called by the matcher once with each document seen by the matcher during the match process. Note that the matcher will often not see all the documents which match the query, due to optimisations which allow low-weighted documents to be skipped, and allow the match process to be terminated early.

#### Parameters:

*doc* The document seen by the match spy.

*wt* The weight of the document.

Implemented in [Xapian::ValueCountMatchSpy](#).

### 7.33.3.6 **virtual std::string Xapian::MatchSpy::serialise () const** [virtual]

Return this object's parameters serialised as a single string.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented in [Xapian::ValueCountMatchSpy](#).

#### 7.33.3.7 `virtual std::string Xapian::MatchSpy::serialise_results () const` [virtual]

Serialise the results of this match spy.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented in [Xapian::ValueCountMatchSpy](#).

#### 7.33.3.8 `virtual MatchSpy* Xapian::MatchSpy::unserialise (const std::string & s, const Registry & context) const` [virtual]

Unserialise parameters.

This method unserialises parameters serialised by the [serialise\(\)](#) method and allocates and returns a new object initialised with them.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented in [Xapian::ValueCountMatchSpy](#).

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

- [xapian/matchspy.h](#)

## 7.34 Xapian::MSet Class Reference

A match set ([MSet](#)).

### Public Types

- typedef [MSetIterator](#) [value\\_type](#)  
*Allow use as an STL container.*
- typedef [MSetIterator](#) [iterator](#)  
*Allow use as an STL container.*
- typedef [MSetIterator](#) [const\\_iterator](#)  
*Allow use as an STL container.*
- typedef [MSetIterator](#) & [reference](#)  
*Allow use as an STL container.*
- typedef [MSetIterator](#) & [const\\_reference](#)  
*Allow use as an STL container.*
- typedef [MSetIterator](#) \* [pointer](#)  
*Allow use as an STL container.*
- typedef [Xapian::doccount\\_diff](#) [difference\\_type](#)  
*Allow use as an STL container.*
- typedef [Xapian::doccount](#) [size\\_type](#)  
*Allow use as an STL container.*

### Public Member Functions

- [MSet](#) ()  
*Create an empty [Xapian::MSet](#).*
- [~MSet](#) ()  
*Destroy a [Xapian::MSet](#).*
- [MSet](#) (const [MSet](#) &other)  
*Copying is allowed (and is cheap).*
- void [operator=](#) (const [MSet](#) &other)  
*Assignment is allowed (and is cheap).*
- void [fetch](#) (const [MSetIterator](#) &begin, const [MSetIterator](#) &end) const

*Fetch the document info for a set of items in the [MSet](#).*

- `void fetch (const MSetIterator &item) const`  
*Fetch the single item specified.*
- `void fetch () const`  
*Fetch all the items in the [MSet](#).*
- `Xapian::percent convert\_to\_percent (Xapian::weight wt) const`  
*This converts the weight supplied to a percentage score.*
- `Xapian::percent convert\_to\_percent (const MSetIterator &it) const`  
*Return the percentage score for a particular item.*
- `Xapian::doccount get\_termfreq (const std::string &name) const`  
*Return the term frequency of the given query term.*
- `Xapian::weight get\_termweight (const std::string &name) const`  
*Return the term weight of the given query term.*
- `Xapian::doccount get\_firstitem () const`  
*The index of the first item in the result which was put into the [MSet](#).*
- `Xapian::doccount get\_matches\_lower\_bound () const`  
*A lower bound on the number of documents in the database which match the query.*
- `Xapian::doccount get\_matches\_estimated () const`  
*An estimate for the number of documents in the database which match the query.*
- `Xapian::doccount get\_matches\_upper\_bound () const`  
*An upper bound on the number of documents in the database which match the query.*
- `Xapian::doccount get\_uncollapsed\_matches\_lower\_bound () const`  
*A lower bound on the number of documents in the database which would match the query if collapsing wasn't used.*
- `Xapian::doccount get\_uncollapsed\_matches\_estimated () const`  
*A estimate of the number of documents in the database which would match the query if collapsing wasn't used.*
- `Xapian::doccount get\_uncollapsed\_matches\_upper\_bound () const`  
*A upper bound on the number of documents in the database which would match the query if collapsing wasn't used.*
- `Xapian::weight get\_max\_possible () const`  
*The maximum possible weight in the [MSet](#).*

- [Xapian::weight get\\_max\\_attained](#) () const  
*The greatest weight which is attained by any document in the database.*
- [Xapian::doccount size](#) () const  
*The number of items in this [MSet](#).*
- [Xapian::doccount max\\_size](#) () const  
*Required to allow use as an STL container.*
- bool [empty](#) () const  
*Test if this [MSet](#) is empty.*
- void [swap](#) ([MSet](#) &other)  
*Swap the [MSet](#) we point to with another.*
- [MSetIterator begin](#) () const  
*Iterator for the items in this [MSet](#).*
- [MSetIterator end](#) () const  
*End iterator corresponding to [begin\(\)](#).*
- [MSetIterator back](#) () const  
*Iterator pointing to the last element of this [MSet](#).*
- [MSetIterator operator\[\]](#) ([Xapian::doccount](#) i) const  
*This returns the document at position i in this [MSet](#) object.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.34.1 Detailed Description

A match set ([MSet](#)).

This class represents (a portion of) the results of a query.

### 7.34.2 Member Function Documentation

#### 7.34.2.1 [Xapian::percent](#) [Xapian::MSet::convert\\_to\\_percent](#) ([Xapian::weight wt](#)) const

This converts the weight supplied to a percentage score.

The return value will be in the range 0 to 100, and will be 0 if and only if the item did not match the query at all.



### 7.34.2.2 void Xapian::MSet::fetch (const MSetIterator & *begin*, const MSetIterator & *end*) const

Fetch the document info for a set of items in the [MSet](#).

This method causes the documents in the range specified by the iterators to be fetched from the database, and cached in the [Xapian::MSet](#) object. This has little effect when performing a search across a local database, but will greatly speed up subsequent access to the document contents when the documents are stored in a remote database.

The iterators must be over this [Xapian::MSet](#) - undefined behaviour will result otherwise.

#### Parameters:

*begin* [MSetIterator](#) for first item to fetch.

*end* [MSetIterator](#) for item after last item to fetch.

### 7.34.2.3 Xapian::doccount Xapian::MSet::get\_firstitem () const

The index of the first item in the result which was put into the [MSet](#).

This corresponds to the parameter "first" specified in [Xapian::Enquire::get\\_mset\(\)](#). A value of 0 corresponds to the highest result being the first item in the [MSet](#).

### 7.34.2.4 Xapian::doccount Xapian::MSet::get\_matches\_estimated () const

An estimate for the number of documents in the database which match the query.

This figure takes into account collapsing of duplicates, and weighting cutoff values.

This value is returned because there is sometimes a request to display such information. However, our experience is that presenting this value to users causes them to worry about the large number of results, rather than how useful those at the top of the result set are, and is thus undesirable.

### 7.34.2.5 Xapian::doccount Xapian::MSet::get\_matches\_lower\_bound () const

A lower bound on the number of documents in the database which match the query.

This figure takes into account collapsing of duplicates, and weighting cutoff values.

This number is usually considerably less than the actual number of documents which match the query.

### 7.34.2.6 Xapian::doccount Xapian::MSet::get\_matches\_upper\_bound () const

An upper bound on the number of documents in the database which match the query.

This figure takes into account collapsing of duplicates, and weighting cutoff values.

This number is usually considerably greater than the actual number of documents which match the query.

#### 7.34.2.7 **Xapian::weight Xapian::MSet::get\_max\_attained () const**

The greatest weight which is attained by any document in the database.

If `firstitem == 0` and the primary ordering is by relevance, this is the weight of the first entry in the [MSet](#).

If no documents are found by the query, this will be 0.

Note that calculation of `max_attained` requires calculation of at least one result item - therefore, if no items were requested when the query was performed (by specifying `maxitems = 0` in [Xapian::Enquire::get\\_mset\(\)](#)), this value will be 0.

#### 7.34.2.8 **Xapian::weight Xapian::MSet::get\_max\_possible () const**

The maximum possible weight in the [MSet](#).

This weight is likely not to be attained in the set of results, but represents an upper bound on the weight which a document could attain for the given query.

#### 7.34.2.9 **Xapian::doccount Xapian::MSet::get\_termfreq (const std::string & tname) const**

Return the term frequency of the given query term.

##### **Parameters:**

*tname* The term to look for.

This is sometimes more efficient than asking the database directly for the term frequency - in particular, if the term was in the query, its frequency will usually be cached in the [MSet](#).

#### 7.34.2.10 **Xapian::weight Xapian::MSet::get\_termweight (const std::string & tname) const**

Return the term weight of the given query term.

##### **Parameters:**

*tname* The term to look for.

##### **Exceptions:**

[Xapian::InvalidArgumentError](#) is thrown if the term was not in the query.

**7.34.2.11 Xapian::doccount Xapian::MSet::max\_size () const** [inline]

Required to allow use as an STL container.

**7.34.2.12 MSetIterator Xapian::MSet::operator[] (Xapian::doccount i) const**

This returns the document at position *i* in this [MSet](#) object.

Note that this is not the same as the document at rank *i* in the query, unless the "first" parameter to [Xapian::Enquire::get\\_mset](#) was 0. Rather, it is the document at rank *i* + first.

In other words, the offset is into the documents represented by this object, not into the set of documents matching the query.

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

- [xapian/enquire.h](#)

## 7.35 Xapian::MSetIterator Class Reference

An iterator pointing to items in an [MSet](#).

### Public Types

- typedef [std::bidirectional\\_iterator\\_tag](#) [iterator\\_category](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::docid](#) [value\\_type](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::doccount\\_diff](#) [difference\\_type](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::docid](#) \* [pointer](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::docid](#) & [reference](#)  
*Allow use as an STL iterator.*

### Public Member Functions

- [MSetIterator](#) ()  
*Create an uninitialised iterator; this cannot be used, but is convenient syntactically.*
- [MSetIterator](#) (const [MSetIterator](#) &other)  
*Copying is allowed (and is cheap).*
- void [operator=](#) (const [MSetIterator](#) &other)  
*Assignment is allowed (and is cheap).*
- [MSetIterator](#) & [operator++](#) ()  
*Advance the iterator.*
- [MSetIterator](#) [operator++](#) (int)  
*Advance the iterator (postfix variant).*
- [MSetIterator](#) & [operator--](#) ()  
*Decrement the iterator.*
- [MSetIterator](#) [operator--](#) (int)  
*Decrement the iterator (postfix variant).*
- [Xapian::docid](#) [operator\\*](#) () const

*Get the document ID for the current position.*

- [Xapian::Document get\\_document \(\)](#) const  
*Get a [Xapian::Document](#) object for the current position.*
- [Xapian::doccount get\\_rank \(\)](#) const  
*Get the rank of the document at the current position.*
- [Xapian::weight get\\_weight \(\)](#) const  
*Get the weight of the document at the current position.*
- [std::string get\\_collapse\\_key \(\)](#) const  
*Get the collapse key for this document.*
- [Xapian::doccount get\\_collapse\\_count \(\)](#) const  
*Get an estimate of the number of documents that have been collapsed into this one.*
- [Xapian::percent get\\_percent \(\)](#) const  
*This returns the weight of the document as a percentage score.*
- [std::string get\\_description \(\)](#) const  
*Return a string describing this object.*

## Friends

- [bool operator== \(const \[MSetIterator\]\(#\) &a, const \[MSetIterator\]\(#\) &b\)](#)  
*Equality test for [MSetIterator](#) objects.*
- [bool operator!= \(const \[MSetIterator\]\(#\) &a, const \[MSetIterator\]\(#\) &b\)](#)  
*Inequality test for [MSetIterator](#) objects.*

### 7.35.1 Detailed Description

An iterator pointing to items in an [MSet](#).

This is used for access to individual results of a match.

### 7.35.2 Member Function Documentation

#### 7.35.2.1 [Xapian::doccount Xapian::MSetIterator::get\\_collapse\\_count \(\)](#) const

Get an estimate of the number of documents that have been collapsed into this one.

The estimate will always be less than or equal to the actual number of other documents satisfying the match criteria with the same collapse key as this document.

This method may return 0 even though there are other documents with the same collapse key which satisfying the match criteria. However if this method returns non-zero, there definitely are other such documents. So this method may be used to inform the user that there are "at least N other matches in this group", or to control whether to offer a "show other documents in this group" feature (but note that it may not offer it in every case where it would show other documents).

#### 7.35.2.2 `Xapian::Document Xapian::MSetIterator::get_document () const`

Get a [Xapian::Document](#) object for the current position.

This method returns a [Xapian::Document](#) object which provides the information about the document pointed to by the [MSetIterator](#).

If the underlying database has suitable support, using this call (rather than asking the database for a document based on its document ID) will enable the system to ensure that the correct data is returned, and that the document has not been deleted or changed since the query was performed.

##### Returns:

A [Xapian::Document](#) object containing the document data.

##### Exceptions:

[Xapian::DocNotFoundError](#) The document specified could not be found in the database.

#### 7.35.2.3 `Xapian::percent Xapian::MSetIterator::get_percent () const`

This returns the weight of the document as a percentage score.

The return value will be an integer in the range 0 to 100: 0 meaning that the item did not match the query at all.

The intention is that the highest weighted document will get 100 if it matches all the weight-contributing terms in the query. However, currently it may get a lower percentage score if you use a [MatchDecider](#) and the sorting is primarily by value. In this case, the percentage for a particular document may vary depending on the first, max\_size, and checkatleast parameters passed to [Enquire::get\\_mset\(\)](#) (this bug is hard to fix without having to apply the [MatchDecider](#) to potentially many more documents, which is potentially costly).

#### 7.35.2.4 `Xapian::doccount Xapian::MSetIterator::get_rank () const` [inline]

Get the rank of the document at the current position.

The rank is the position that this document is at in the ordered list of results of the query. The result is 0-based - i.e. the top-ranked document has a rank of 0.

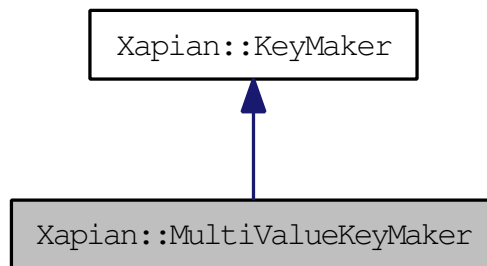
The documentation for this class was generated from the following file:

- [xapian/enquire.h](#)

## 7.36 Xapian::MultiValueKeyMaker Class Reference

[KeyMaker](#) subclass which combines several values.

Inheritance diagram for Xapian::MultiValueKeyMaker:



### Public Member Functions

- virtual std::string [operator\(\)](#) (const [Xapian::Document](#) &doc) const

*This method takes a [Document](#) object and builds a key string from it.*

#### 7.36.1 Detailed Description

[KeyMaker](#) subclass which combines several values.

When the result is used for sorting, results are ordered by the first value. In the event of a tie, the second is used. If this is the same for both, the third is used, and so on. If *reverse* is true for a value, then the sort order for that value is reversed.

When used for collapsing, the documents will only be considered equal if all the values specified match. If none of the specified values are set then the generated key will be empty, so such documents won't be collapsed (which is consistent with the behaviour in the "collapse on a value" case). If you'd prefer that documents with none of the keys set are collapsed together, then you can set *reverse* for at least one of the values. Other than this, it isn't useful to set *reverse* for collapsing.

### 7.36.2 Member Function Documentation

#### 7.36.2.1 virtual std::string Xapian::MultiValueKeyMaker::operator() (const Xapian::Document & doc) const [virtual]

This method takes a [Document](#) object and builds a key string from it.

These keys are then used for ordering or collapsing matching documents.

Implements [Xapian::KeyMaker](#).

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

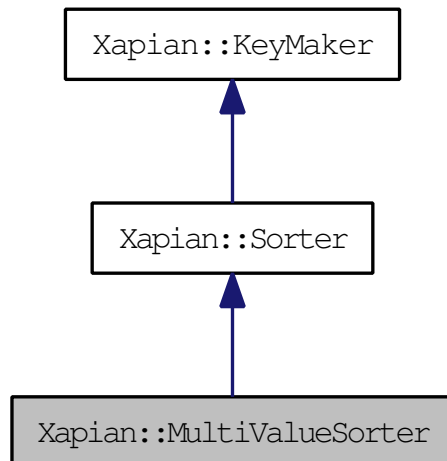


- [xapian/keymaker.h](#)

## 7.37 Xapian::MultiValueSorter Class Reference

[Sorter](#) subclass which sorts by a several values.

Inheritance diagram for Xapian::MultiValueSorter:



### Public Member Functions

- virtual std::string [operator\(\)](#) (const [Xapian::Document](#) &doc) const  
*This method takes a [Document](#) object and builds a key string from it.*

#### 7.37.1 Detailed Description

[Sorter](#) subclass which sorts by a several values.

Results are ordered by the first value. In the event of a tie, the second is used. If this is the same for both, the third is used, and so on.

#### Deprecated

This class is deprecated - you should migrate to using [MultiValueKeyMaker](#) instead. Note that `MultiValueSorter::add()` becomes `MultiValueKeyMaker::add_value()`, but the sense of the direction flag is reversed (to be consistent with [Enquire::set\\_sort\\_by\\_value\(\)](#)), so:

```
MultiValueSorter sorter; // Primary ordering is forwards on value 4. sorter.add(4); //  
Secondary ordering is reverse on value 5. sorter.add(5, false);
```

becomes:

```
MultiValueKeyMaker sorter; // Primary ordering is forwards on value 4. sorter.add_  
value(4); // Secondary ordering is reverse on value 5. sorter.add_value(5, true);
```

## 7.37.2 Member Function Documentation

### 7.37.2.1 `virtual std::string Xapian::MultiValueSorter::operator() (const Xapian::Document & doc) const` [virtual]

This method takes a [Document](#) object and builds a key string from it.

These keys are then used for ordering or collapsing matching documents.

Implements [Xapian::KeyMaker](#).

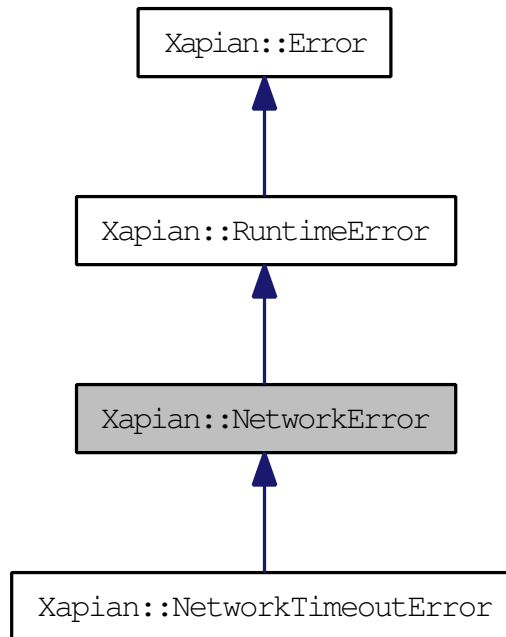
The documentation for this class was generated from the following file:

- [xapian/keymaker.h](#)

## 7.38 Xapian::NetworkError Class Reference

Indicates a problem communicating with a remote database.

Inheritance diagram for Xapian::NetworkError:



### Public Member Functions

- [NetworkError](#) (const std::string &msg\_, const std::string &context\_ =std::string(), int errno\_=0)

*General purpose constructor which allows setting errno.*

- [NetworkError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.38.1 Detailed Description

Indicates a problem communicating with a remote database.

## 7.38.2 Constructor & Destructor Documentation

**7.38.2.1 Xapian::NetworkError::NetworkError** (const std::string & *msg\_*,  
const std::string & *context\_* = std::string(), int *errno\_* = 0)  
[inline, explicit]

General purpose constructor which allows setting errno.

**7.38.2.2 Xapian::NetworkError::NetworkError** (const std::string & *msg\_*, int  
*errno\_*) [inline]

Construct from message and errno value.

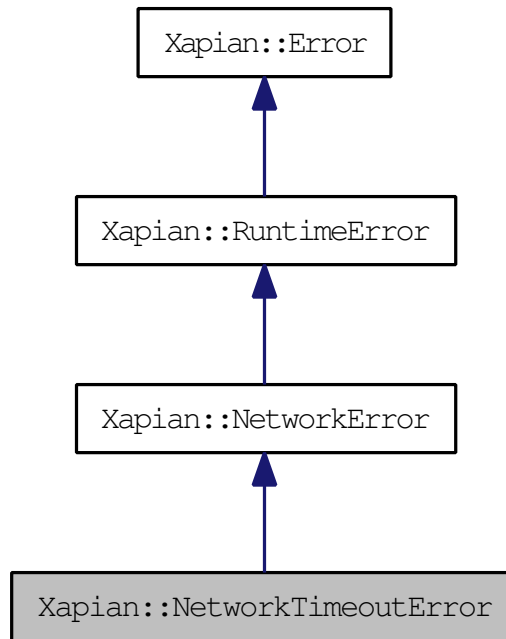
The documentation for this class was generated from the following file:

- xapian/[error.h](#)

## 7.39 Xapian::NetworkTimeoutError Class Reference

Indicates a timeout expired while communicating with a remote database.

Inheritance diagram for Xapian::NetworkTimeoutError:



### Public Member Functions

- [NetworkTimeoutError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [NetworkTimeoutError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.39.1 Detailed Description

Indicates a timeout expired while communicating with a remote database.

## 7.39.2 Constructor & Destructor Documentation

**7.39.2.1 Xapian::NetworkTimeoutError::NetworkTimeoutError (const std::string & msg\_, const std::string & context\_ = std::string(), int errno\_ = 0) [inline, explicit]**

General purpose constructor which allows setting errno.

**7.39.2.2 Xapian::NetworkTimeoutError::NetworkTimeoutError (const std::string & msg\_, int errno\_) [inline]**

Construct from message and errno value.

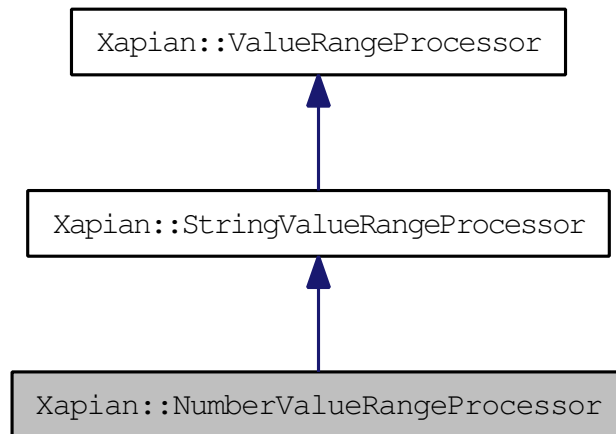
The documentation for this class was generated from the following file:

- [xapian/error.h](#)

## 7.40 Xapian::NumberValueRangeProcessor Class Reference

Handle a number range.

Inheritance diagram for Xapian::NumberValueRangeProcessor:



### Public Member Functions

- [NumberValueRangeProcessor](#) ([Xapian::valueno](#) slot\_)

*Constructor.*

- [NumberValueRangeProcessor](#) ([Xapian::valueno](#) slot\_, const std::string &str\_, bool prefix\_=true)

*Constructor.*

- [Xapian::valueno operator\(\)](#) (std::string &begin, std::string &end)

*Check for a valid range of this type.*

### 7.40.1 Detailed Description

Handle a number range.

This class must be used on values which have been encoded using [Xapian::sortable\\_-serialise\(\)](#) which turns numbers into strings which will sort in the same order as the numbers (the same values can be used to implement a numeric sort).



## 7.40.2 Constructor & Destructor Documentation

### 7.40.2.1 Xapian::NumberValueRangeProcessor::NumberValueRangeProcessor (Xapian::valueno *slot\_*) [inline]

Constructor.

**Parameters:**

*slot\_* The value number to return from operator().

### 7.40.2.2 Xapian::NumberValueRangeProcessor::NumberValueRangeProcessor (Xapian::valueno *slot\_*, const std::string & *str\_*, bool *prefix\_* = true) [inline]

Constructor.

**Parameters:**

*slot\_* The value number to return from operator().

*str\_* A string to look for to recognise values as belonging to this numeric range.

*prefix\_* Whether to look for the string at the start or end of the values. If true, the string is a prefix; if false, the string is a suffix (default: true).

The string supplied in *str\_* is used by *operator()* to decide whether the pair of strings supplied to it constitute a valid range. If *prefix\_* is true, the first value in a range must begin with *str\_* (and the second value may optionally begin with *str\_*); if *prefix\_* is false, the second value in a range must end with *str\_* (and the first value may optionally end with *str\_*).

If *str\_* is empty, the setting of *prefix\_* is irrelevant, and no special strings are required at the start or end of the strings defining the range.

The remainder of both strings defining the endpoints must be valid floating point numbers. (FIXME: define format recognised).

For example, if *str\_* is "\$" and *prefix\_* is true, and the range processor has been added to the queryparser, the queryparser will accept "\$10..50" or "\$10..\$50", but not "10..50" or "10..\$50" as valid ranges. If *str\_* is "kg" and *prefix\_* is false, the queryparser will accept "10..50kg" or "10kg..50kg", but not "10..50" or "10kg..50" as valid ranges.

## 7.40.3 Member Function Documentation

### 7.40.3.1 Xapian::valueno Xapian::NumberValueRangeProcessor::operator() (std::string & *begin*, std::string & *end*) [virtual]

Check for a valid range of this type.

If BEGIN..END is a valid numeric value range, and has the appropriate prefix or suffix (if specified) required for this [NumberValueRangeProcessor](#), this method returns the

value number of range filter on, and sets begin and end to the appropriate serialised values needed to delimit the range. Otherwise it returns [Xapian::BAD\\_VALUENO](#).

Reimplemented from [Xapian::StringValueRangeProcessor](#).

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

- [xapian/queryparser.h](#)

## 7.41 Xapian::PositionIterator Class Reference

An iterator pointing to items in a list of positions.

### Public Member Functions

- [PositionIterator](#) ()  
*Default constructor - for declaring an uninitialised iterator.*
- [~PositionIterator](#) ()  
*Destructor.*
- [PositionIterator](#) (const [PositionIterator](#) &o)  
*Copying is allowed.*
- void [operator=](#) (const [PositionIterator](#) &o)  
*Assignment is allowed.*
- void [skip\\_to](#) ([Xapian::termpos](#) pos)  
*Advance the iterator to the specified termpos.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### Friends

- bool [operator==](#) (const [PositionIterator](#) &a, const [PositionIterator](#) &b)  
*Test equality of two PositionIterators.*

#### 7.41.1 Detailed Description

An iterator pointing to items in a list of positions.

#### 7.41.2 Constructor & Destructor Documentation

##### 7.41.2.1 Xapian::PositionIterator::PositionIterator (const PositionIterator & o)

Copying is allowed.

The internals are reference counted, so copying is also cheap.

### 7.41.3 Member Function Documentation

#### 7.41.3.1 `void Xapian::PositionIterator::operator= (const PositionIterator & o)`

Assignment is allowed.

The internals are reference counted, so assignment is also cheap.

#### 7.41.3.2 `void Xapian::PositionIterator::skip_to (Xapian::termpos pos)`

Advance the iterator to the specified termpos.

If the specified termpos isn't in the list, position ourselves on the first termpos after it (or `at_end()` if no greater term positions are present).

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

- `xapian/`[positioniterator.h](#)

## 7.42 Xapian::PostingIterator Class Reference

An iterator pointing to items in a list of postings.

### Public Types

- typedef std::input\_iterator\_tag [iterator\\_category](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::docid](#) [value\\_type](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::doccount\\_diff](#) [difference\\_type](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::docid](#) \* [pointer](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::docid](#) & [reference](#)  
*Allow use as an STL iterator.*

### Public Member Functions

- [PostingIterator](#) ()  
*Default constructor - for declaring an uninitialised iterator.*
- [~PostingIterator](#) ()  
*Destructor.*
- [PostingIterator](#) (const [PostingIterator](#) &other)  
*Copying is allowed.*
- void [operator=](#) (const [PostingIterator](#) &other)  
*Assignment is allowed.*
- void [skip\\_to](#) ([Xapian::docid](#) did)  
*Advance the iterator to the specified docid.*
- [Xapian::docid](#) [operator\\*](#) () const  
*Get the document id at the current position in the postlist.*
- [Xapian::termcount](#) [get\\_doclength](#) () const  
*Get the length of the document at the current position in the postlist.*
- [Xapian::termcount](#) [get\\_wdf](#) () const

*Get the within document frequency of the document at the current position in the postlist.*

- [PositionIterator positionlist\\_begin \(\) const](#)  
*Return [PositionIterator](#) pointing to start of positionlist for current document.*
- [PositionIterator positionlist\\_end \(\) const](#)  
*Return [PositionIterator](#) pointing to end of positionlist for current document.*
- `std::string get\_description \(\) const`  
*Return a string describing this object.*

## Friends

- `bool operator== (const PostingIterator &a, const PostingIterator &b)`  
*Test equality of two [PostingIterators](#).*

### 7.42.1 Detailed Description

An iterator pointing to items in a list of postings.

### 7.42.2 Constructor & Destructor Documentation

#### 7.42.2.1 `Xapian::PostingIterator::PostingIterator (const PostingIterator &other)`

Copying is allowed.

The internals are reference counted, so copying is also cheap.

### 7.42.3 Member Function Documentation

#### 7.42.3.1 `Xapian::termcount Xapian::PostingIterator::get_doclength () const`

Get the length of the document at the current position in the postlist.

This information may be stored in the postlist, in which case this lookup should be extremely fast (indeed, not require further disk access). If the information is not present in the postlist, it will be retrieved from the database, at a greater performance cost.

#### 7.42.3.2 `void Xapian::PostingIterator::operator= (const PostingIterator &other)`

Assignment is allowed.

The internals are reference counted, so assignment is also cheap.

#### 7.42.3.3 void Xapian::PostingIterator::skip\_to (Xapian::docid *did*)

Advance the iterator to the specified docid.

If the specified docid isn't in the list, position ourselves on the first document after it (or `at_end()` if no greater docids are present).

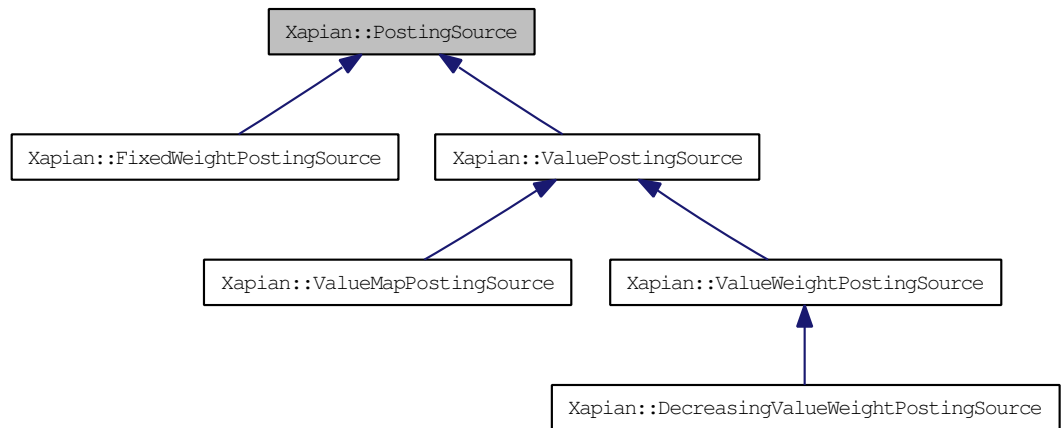
The documentation for this class was generated from the following file:

- `xapian/postingiterator.h`

## 7.43 Xapian::PostingSource Class Reference

Base class which provides an "external" source of postings.

Inheritance diagram for Xapian::PostingSource:



### Public Member Functions

- virtual [Xapian::doccount get\\_termfreq\\_min \(\)](#) const =0  
*A lower bound on the number of documents this object can return.*
- virtual [Xapian::doccount get\\_termfreq\\_est \(\)](#) const =0  
*An estimate of the number of documents this object can return.*
- virtual [Xapian::doccount get\\_termfreq\\_max \(\)](#) const =0  
*An upper bound on the number of documents this object can return.*
- [Xapian::weight get\\_maxweight \(\)](#) const  
*Return the currently set upper bound on what [get\\_weight\(\)](#) can return.*
- virtual [Xapian::weight get\\_weight \(\)](#) const  
*Return the weight contribution for the current document.*
- virtual [Xapian::docid get\\_docid \(\)](#) const =0  
*Return the current docid.*
- virtual void [next \(Xapian::weight min\\_wt\)=0](#)  
*Advance the current position to the next matching document.*
- virtual void [skip\\_to \(Xapian::docid did, Xapian::weight min\\_wt\)](#)  
*Advance to the specified docid.*



- virtual bool [check](#) ([Xapian::docid](#) did, [Xapian::weight](#) min\_wt)  
*Check if the specified docid occurs.*
- virtual bool [at\\_end](#) () const =0  
*Return true if the current position is past the last entry in this list.*
- virtual [PostingSource](#) \* [clone](#) () const  
*Clone the posting source.*
- virtual std::string [name](#) () const  
*Name of the posting source class.*
- virtual std::string [serialise](#) () const  
*Serialise object parameters into a string.*
- virtual [PostingSource](#) \* [unserialise](#) (const std::string &s) const  
*Create object given string serialisation returned by [serialise\(\)](#).*
- virtual void [init](#) (const [Database](#) &db)=0  
*Set this [PostingSource](#) to the start of the list of postings.*
- virtual std::string [get\\_description](#) () const  
*Return a string describing this object.*

## Protected Member Functions

- [PostingSource](#) ()  
*Allow subclasses to be instantiated.*
- void [set\\_maxweight](#) ([Xapian::weight](#) max\_weight)  
*Set an upper bound on what [get\\_weight\(\)](#) can return from now on.*

### 7.43.1 Detailed Description

Base class which provides an "external" source of postings.

### 7.43.2 Member Function Documentation

#### 7.43.2.1 virtual bool Xapian::PostingSource::at\_end () const [pure virtual]

Return true if the current position is past the last entry in this list.

At least one of [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#) will be called before this method is first called.

Implemented in [Xapian::ValuePostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.2 **virtual bool Xapian::PostingSource::check (Xapian::docid *did*, Xapian::weight *min\_wt*)** [virtual]

Check if the specified docid occurs.

The caller is required to ensure that the specified document id *did* actually exists in the database. If it does, it must move to that document id, and return true. If it does not, it may either:

- return true, having moved to a definite position (including "at\_end"), which must be the same position as [skip\\_to\(\)](#) would have moved to.

or

- return false, having moved to an "indeterminate" position, such that a subsequent call to [next\(\)](#) or [skip\\_to\(\)](#) will move to the next matching position after *did*.

Generally, this method should act like [skip\\_to\(\)](#) and return true if that can be done at little extra cost.

Otherwise it should simply check if a particular docid is present, returning true if it is, and false if it isn't.

The default implementation calls [skip\\_to\(\)](#) and always returns true.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

Reimplemented in [Xapian::ValuePostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.3 **virtual PostingSource\* Xapian::PostingSource::clone ()** const [virtual]

Clone the posting source.

The clone should inherit the configuration of the parent, but need not inherit the state. ie, the clone does not need to be in the same iteration position as the original: the matcher will always call [init\(\)](#) on the clone before attempting to move the iterator, or read the information about the current position of the iterator.

This may return NULL to indicate that cloning is not supported. In this case, the [PostingSource](#) may only be used with a single-database search.

The default implementation returns NULL.

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented in [Xapian::ValueWeightPostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), [Xapian::ValueMapPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.4 `virtual std::string Xapian::PostingSource::get_description () const` [virtual]

Return a string describing this object.

This default implementation returns a generic answer. This default is provided to avoid forcing those deriving their own [PostingSource](#) subclass from having to implement this (they may not care what [get\\_description\(\)](#) gives for their subclass).

Reimplemented in [Xapian::ValueWeightPostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), [Xapian::ValueMapPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.5 `virtual Xapian::docid Xapian::PostingSource::get_docid () const` [pure virtual]

Return the current docid.

This method may assume that it will only be called when there is a "current document". See [get\\_weight\(\)](#) for details.

Note: in the case of a multi-database search, the returned docid should be in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

Implemented in [Xapian::ValuePostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.6 `virtual Xapian::doccount Xapian::PostingSource::get_termfreq_est () const` [pure virtual]

An estimate of the number of documents this object can return.

It must always be true that:

[get\\_termfreq\\_min\(\)](#) <= [get\\_termfreq\\_est\(\)](#) <= [get\\_termfreq\\_max\(\)](#)

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implemented in [Xapian::ValuePostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.7 `virtual Xapian::doccount Xapian::PostingSource::get_termfreq_max () const` [pure virtual]

An upper bound on the number of documents this object can return.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implemented in [Xapian::ValuePostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.8 **virtual Xapian::doccount Xapian::PostingSource::get\_termfreq\_min () const** [pure virtual]

A lower bound on the number of documents this object can return.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implemented in [Xapian::ValuePostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.9 **virtual Xapian::weight Xapian::PostingSource::get\_weight () const** [virtual]

Return the weight contribution for the current document.

This default implementation always returns 0, for convenience when implementing "weight-less" [PostingSource](#) subclasses.

This method may assume that it will only be called when there is a "current document". In detail: [Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time. It will also only call this if the [PostingSource](#) reports that it is pointing to a valid document (ie, it will not call it before calling at least one of [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#), and will ensure that the [PostingSource](#) is not at the end by calling [at\\_end\(\)](#)).

Reimplemented in [Xapian::ValueWeightPostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), [Xapian::ValueMapPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.10 **virtual void Xapian::PostingSource::init (const Database & db)** [pure virtual]

Set this [PostingSource](#) to the start of the list of postings.

This is called automatically by the matcher prior to each query being processed.

If a [PostingSource](#) is used for multiple searches, [init\(\)](#) will therefore be called multiple times, and must handle this by using the database passed in the most recent call.

#### Parameters:

*db* The database which the [PostingSource](#) should iterate through.

Note: the database supplied to this method must not be modified: in particular, the [reopen\(\)](#) method should not be called on it.

Note: in the case of a multi-database search, a separate [PostingSource](#) will be used for each database (the separate PostingSources will be obtained using [clone\(\)](#)), and each

[PostingSource](#) will be passed one of the sub-databases as the *db* parameter here. The *db* parameter will therefore always refer to a single database. All docids passed to, or returned from, the [PostingSource](#) refer to docids in that single database, rather than in the multi-database.

Implemented in [Xapian::ValuePostingSource](#), [Xapian::ValueWeightPostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), [Xapian::ValueMapPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.11 `virtual std::string Xapian::PostingSource::name () const` [virtual]

Name of the posting source class.

This is used when serialising and unserialising posting sources; for example, for performing remote searches.

If the subclass is in a C++ namespace, the namespace should be included in the name, using "::" as a separator. For example, for a [PostingSource](#) subclass called "FooPostingSource" in the "Xapian" namespace the result of this call should be "Xapian::FooPostingSource".

This should only be implemented if [serialise\(\)](#) and [unserialise\(\)](#) are also implemented. The default implementation returns an empty string.

If this returns an empty string, [Xapian](#) will assume that [serialise\(\)](#) and [unserialise\(\)](#) are not implemented.

Reimplemented in [Xapian::ValueWeightPostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), [Xapian::ValueMapPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.12 `virtual void Xapian::PostingSource::next (Xapian::weight min_wt)` [pure virtual]

Advance the current position to the next matching document.

The [PostingSource](#) starts before the first entry in the list, so [next\(\)](#) must be called before any methods which need the context of the current position.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

#### Parameters:

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Implemented in [Xapian::ValuePostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.13 **virtual std::string Xapian::PostingSource::serialise () const** [virtual]

Serialise object parameters into a string.

The serialised parameters should represent the configuration of the posting source, but need not (indeed, should not) represent the current iteration state.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented in [Xapian::ValueWeightPostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), [Xapian::ValueMapPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

#### 7.43.2.14 **void Xapian::PostingSource::set\_maxweight (Xapian::weight max\_weight)** [protected]

Set an upper bound on what [get\\_weight\(\)](#) can return from now on.

This upper bound is used by the matcher to perform various optimisations, so if you can return a good bound, then matches will generally run faster.

This method should be called after calling [init\(\)](#), and may be called during iteration if the upper bound drops.

It is valid for the posting source to have returned a higher value from [get\\_weight\(\)](#) earlier in the iteration, but the posting source must not return a higher value from [get\\_weight\(\)](#) than the currently set upper bound, and the upper bound must not be increased (until [init\(\)](#) has been called).

If you don't call this method, the upper bound will default to 0, for convenience when implementing "weight-less" [PostingSource](#) subclasses.

#### 7.43.2.15 **virtual void Xapian::PostingSource::skip\_to (Xapian::docid did, Xapian::weight min\_wt)** [virtual]

Advance to the specified docid.

If the specified docid isn't in the list, position ourselves on the first document after it (or [at\\_end\(\)](#) if no greater docids are present).

If the current position is already the specified docid, this method will leave the position unmodified.

If the specified docid is earlier than the current position, the behaviour is unspecified. A sensible behaviour would be to leave the current position unmodified, but it is also reasonable to move to the specified docid.

The default implementation calls [next\(\)](#) repeatedly, which works but [skip\\_to\(\)](#) can often be implemented much more efficiently.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the

single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

**Parameters:**

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Reimplemented in [Xapian::ValuePostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

**7.43.2.16 virtual PostingSource\* Xapian::PostingSource::unserialise (const std::string & s) const [virtual]**

Create object given string serialisation returned by [serialise\(\)](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

**Parameters:**

*s* A serialised instance of this [PostingSource](#) subclass.

Reimplemented in [Xapian::ValueWeightPostingSource](#), [Xapian::DecreasingValueWeightPostingSource](#), [Xapian::ValueMapPostingSource](#), and [Xapian::FixedWeightPostingSource](#).

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

- [xapian/postingsource.h](#)

## 7.44 Xapian::Query Class Reference

Class representing a query.

### Public Types

- enum `op` {  
`OP_AND`, `OP_OR`, `OP_AND_NOT`, `OP_XOR`,  
`OP_AND_MAYBE`, `OP_FILTER`, `OP_NEAR`, `OP_PHRASE`,  
`OP_VALUE_RANGE`, `OP_SCALE_WEIGHT`, `OP_ELITE_SET`, `OP_VALUE_GE`,  
`OP_VALUE_LE`, `OP_SYNONYM` }

*Enum of possible query operations.*

### Public Member Functions

- `Query` (const `Query` &copyme)  
*Copy constructor.*
- `Query` & `operator=` (const `Query` &copyme)  
*Assignment.*
- `Query` ()  
*Default constructor: makes an empty query which matches no documents.*
- `~Query` ()  
*Destructor.*
- `Query` (const std::string &name\_, `Xapian::termcount` wqf\_=1, `Xapian::termpos` pos\_=0)  
*A query consisting of a single term.*
- `Query` (`Query::op` op\_, const `Query` &left, const `Query` &right)  
*A query consisting of two subqueries, opp-ed together.*
- `Query` (`Query::op` op\_, const std::string &left, const std::string &right)  
*A query consisting of two termnames opp-ed together.*
- template<class Iterator >  
`Query` (`Query::op` op\_, Iterator qbegin, Iterator qend, `Xapian::termcount` parameter=0)  
*Combine a number of `Xapian::Query`-s with the specified operator.*
- `Query` (`Query::op` op\_, `Xapian::Query` q, double parameter)



Apply the specified operator to a single [Xapian::Query](#) object, with a double parameter.

- [Query](#) ([Query::op](#) op\_, [Xapian::valueno](#) slot, const std::string &begin, const std::string &end)  
Construct a value range query on a document value.
- [Query](#) ([Query::op](#) op\_, [Xapian::valueno](#) slot, const std::string &value)  
Construct a value comparison query on a document value.
- [Query](#) ([Xapian::PostingSource](#) \*external\_source)  
Construct an external source query.
- [Xapian::termcount](#) get\_length () const  
Get the length of the query, used by some ranking formulae.
- [TermIterator](#) get\_terms\_begin () const  
Return a [Xapian::TermIterator](#) returning all the terms in the query, in order of termpos.
- [TermIterator](#) get\_terms\_end () const  
Return a [Xapian::TermIterator](#) to the end of the list of terms in the query.
- bool empty () const  
Test if the query is empty (i.e.
- std::string serialise () const  
Serialise query into a string.
- std::string get\_description () const  
Return a string describing this object.

## Static Public Member Functions

- static [Query](#) unserialise (const std::string &s)  
Unserialise a query from a string produced by [serialise\(\)](#).
- static [Query](#) unserialise (const std::string &s, const [Registry](#) &registry)  
Unserialise a query from a string produced by [serialise\(\)](#).

## Static Public Attributes

- static const [Xapian::Query](#) MatchAll  
A query which matches all documents in the database.

- static const [Xapian::Query MatchNothing](#)

*A query which matches no documents.*

### 7.44.1 Detailed Description

Class representing a query.

Queries are represented as a tree of objects.

### 7.44.2 Member Enumeration Documentation

#### 7.44.2.1 enum Xapian::Query::op

Enum of possible query operations.

##### Enumerator:

**OP\_AND** Return iff both subqueries are satisfied.

**OP\_OR** Return if either subquery is satisfied.

**OP\_AND\_NOT** Return if left but not right satisfied.

**OP\_XOR** Return if one query satisfied, but not both.

**OP\_AND\_MAYBE** Return iff left satisfied, but use weights from both.

**OP\_FILTER** As AND, but use only weights from left subquery.

**OP\_NEAR** Find occurrences of a list of terms with all the terms occurring within a specified window of positions.

Each occurrence of a term must be at a different position, but the order they appear in is irrelevant.

The window parameter should be specified for this operation, but will default to the number of terms in the list.

**OP\_PHRASE** Find occurrences of a list of terms with all the terms occurring within a specified window of positions, and all the terms appearing in the order specified.

Each occurrence of a term must be at a different position.

The window parameter should be specified for this operation, but will default to the number of terms in the list.

**OP\_VALUE\_RANGE** Filter by a range test on a document value.

**OP\_SCALE\_WEIGHT** Scale the weight of a subquery by the specified factor.

A factor of 0 means this subquery will contribute no weight to the query - it will act as a purely boolean subquery.

If the factor is negative, [Xapian::InvalidArgumentError](#) will be thrown.

**OP\_ELITE\_SET** Select an elite set from the subqueries, and perform a query with these combined as an OR query.

**OP\_VALUE\_GE** Filter by a greater-than-or-equal test on a document value.

**OP\_VALUE\_LE** Filter by a less-than-or-equal test on a document value.

**OP\_SYNONYM** Treat a set of queries as synonyms.

This returns all results which match at least one of the queries, but weighting as if all the sub-queries are instances of the same term: so multiple matching terms for a document increase the wdf value used, and the term frequency is based on the number of documents which would match an OR of all the subqueries.

The term frequency used will usually be an approximation, because calculating the precise combined term frequency would be overly expensive.

Identical to OP\_OR, except for the weightings returned.

### 7.44.3 Constructor & Destructor Documentation

#### 7.44.3.1 Xapian::Query::Query (const Query & *copyme*)

Copy constructor.

#### 7.44.3.2 Xapian::Query::Query ()

Default constructor: makes an empty query which matches no documents.

Also useful for defining a [Query](#) object to be assigned to later.

An exception will be thrown if an attempt is made to use an undefined query when building up a composite query.

#### 7.44.3.3 Xapian::Query::~~Query ()

Destructor.

#### 7.44.3.4 Xapian::Query::Query (const std::string & *tname*\_, Xapian::termcount *wqf*\_ = 1, Xapian::termpos *pos*\_ = 0)

A query consisting of a single term.

#### 7.44.3.5 Xapian::Query::Query (Query::op *op*\_, const Query & *left*, const Query & *right*)

A query consisting of two subqueries, opp-ed together.

#### 7.44.3.6 Xapian::Query::Query (Query::op *op*\_, const std::string & *left*, const std::string & *right*)

A query consisting of two termnames opp-ed together.

**7.44.3.7** `template<class Iterator > Xapian::Query::Query (Query::op op_,  
Iterator qbegin, Iterator qend, Xapian::termcount parameter = 0)  
[inline]`

Combine a number of [Xapian::Query](#)-s with the specified operator.

The [Xapian::Query](#) objects are specified with begin and end iterators.

AND, OR, SYNONYM, NEAR and PHRASE can take any number of subqueries. Other operators take exactly two subqueries.

The iterators may be to [Xapian::Query](#) objects, pointers to [Xapian::Query](#) objects, or termnames (std::string-s).

For NEAR and PHRASE, a window size can be specified in parameter.

For ELITE\_SET, the elite set size can be specified in parameter.

**7.44.3.8** `Xapian::Query::Query (Query::op op_, Xapian::valueno slot, const  
std::string & begin, const std::string & end)`

Construct a value range query on a document value.

A value range query matches those documents which have a value stored in the slot given by *slot* which is in the range specified by *begin* and *end* (in lexicographical order), including the endpoints.

#### Parameters:

*op\_* The operator to use for the query. Currently, must be OP\_VALUE\_RANGE.

*slot* The slot number to get the value from.

*begin* The start of the range.

*end* The end of the range.

**7.44.3.9** `Xapian::Query::Query (Query::op op_, Xapian::valueno slot, const  
std::string & value)`

Construct a value comparison query on a document value.

This query matches those documents which have a value stored in the slot given by *slot* which compares, as specified by the operator, to *value*.

#### Parameters:

*op\_* The operator to use for the query. Currently, must be OP\_VALUE\_GE or OP\_VALUE\_LE.

*slot* The slot number to get the value from.

*value* The value to compare.

#### 7.44.3.10 Xapian::Query::Query (Xapian::PostingSource \* *external\_source*) [explicit]

Construct an external source query.

An attempt to clone the posting source will be made immediately, so if the posting source supports clone(), the source supplied may be safely deallocated after this call. If the source does not support clone(), the caller must ensure that the posting source remains valid until the [Query](#) is deallocated.

##### Parameters:

*external\_source* The source to use in the query.

### 7.44.4 Member Function Documentation

#### 7.44.4.1 bool Xapian::Query::empty () const

Test if the query is empty (i.e.

was constructed using the default ctor or with an empty iterator ctor).

#### 7.44.4.2 Xapian::termcount Xapian::Query::get\_length () const

Get the length of the query, used by some ranking formulae.

This value is calculated automatically - if you want to override it you can pass a different value to [Enquire::set\\_query\(\)](#).

#### 7.44.4.3 TermIterator Xapian::Query::get\_terms\_begin () const

Return a [Xapian::TermIterator](#) returning all the terms in the query, in order of termpos.

If multiple terms have the same term position, their order is unspecified. Duplicates (same term and termpos) will be removed.

#### 7.44.4.4 Query& Xapian::Query::operator= (const Query & *copyme*)

Assignment.

#### 7.44.4.5 std::string Xapian::Query::serialise () const

Serialise query into a string.

The query representation may change between [Xapian](#) releases: even between minor versions. However, it is guaranteed not to change unless the remote database protocol has also changed between releases.

#### 7.44.4.6 static Query Xapian::Query::unserialise (const std::string & *s*, const Registry & *registry*) [static]

Unserialise a query from a string produced by [serialise\(\)](#).

The supplied registry will be used to attempt to unserialise any external [PostingSource](#) leaf nodes. This method will fail if the query contains any external [PostingSource](#) leaf nodes which are not registered in the registry.

##### Parameters:

- s* The string representing the serialised query.
- registry* [Xapian::Registry](#) to use.

#### 7.44.4.7 static Query Xapian::Query::unserialise (const std::string & *s*) [static]

Unserialise a query from a string produced by [serialise\(\)](#).

This method will fail if the query contains any external [PostingSource](#) leaf nodes.

##### Parameters:

- s* The string representing the serialised query.

### 7.44.5 Member Data Documentation

#### 7.44.5.1 const Xapian::Query Xapian::Query::MatchAll [static]

A query which matches all documents in the database.

#### 7.44.5.2 const Xapian::Query Xapian::Query::MatchNothing [static]

A query which matches no documents.

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

- [xapian/query.h](#)

## 7.45 Xapian::QueryParser Class Reference

Build a [Xapian::Query](#) object from a user query string.

### Public Types

- enum [feature\\_flag](#) {  
    [FLAG\\_BOOLEAN](#) = 1, [FLAG\\_PHRASE](#) = 2, [FLAG\\_LOVEHATE](#) = 4,  
    [FLAG\\_BOOLEAN\\_ANY\\_CASE](#) = 8,  
    [FLAG\\_WILDCARD](#) = 16, [FLAG\\_PURE\\_NOT](#) = 32, [FLAG\\_PARTIAL](#) = 64,  
    [FLAG\\_SPELLING\\_CORRECTION](#) = 128,  
    [FLAG\\_SYNONYM](#) = 256, [FLAG\\_AUTO\\_SYNONYMS](#) = 512, [FLAG\\_-](#)  
    [AUTO\\_MULTIWORD\\_SYNONYMS](#) = 1024 | [FLAG\\_AUTO\\_SYNONYMS](#),  
    [FLAG\\_DEFAULT](#) = [FLAG\\_PHRASE](#)|[FLAG\\_BOOLEAN](#)|[FLAG\\_LOVEHATE](#)  
}

*Enum of feature flags.*

### Public Member Functions

- [QueryParser](#) (const [QueryParser](#) &o)  
*Copy constructor.*
- [QueryParser](#) & [operator=](#) (const [QueryParser](#) &o)  
*Assignment.*
- [QueryParser](#) ()  
*Default constructor.*
- [~QueryParser](#) ()  
*Destructor.*
- void [set\\_stemmer](#) (const [Xapian::Stem](#) &stemmer)  
*Set the stemmer.*
- void [set\\_stemming\\_strategy](#) (stem\_strategy strategy)  
*Set the stemming strategy.*
- void [set\\_stopper](#) (const [Stopper](#) \*stop=NULL)  
*Set the stopper.*
- void [set\\_default\\_op](#) ([Query::op](#) default\_op)  
*Set the default operator.*
- [Query::op](#) [get\\_default\\_op](#) () const

*Get the current default operator.*

- void [set\\_database](#) (const [Database](#) &db)  
*Specify the database being searched.*
- void [set\\_max\\_wildcard\\_expansion](#) ([Xapian::termcount](#) limit)  
*Specify the maximum expansion of a wildcard term.*
- [Query parse\\_query](#) (const std::string &query\_string, unsigned flags=FLAG\_DEFAULT, const std::string &default\_prefix=std::string())  
*Parse a query.*
- void [add\\_prefix](#) (const std::string &field, const std::string &prefix)  
*Add a probabilistic term prefix.*
- void [add\\_boolean\\_prefix](#) (const std::string &field, const std::string &prefix, bool exclusive)  
*Add a boolean term prefix allowing the user to restrict a search with a boolean filter specified in the free text query.*
- [TermIterator stoplist\\_begin](#) () const  
*Iterate over terms omitted from the query as stopwords.*
- [TermIterator unstem\\_begin](#) (const std::string &term) const  
*Iterate over unstemmed forms of the given (stemmed) term used in the query.*
- void [add\\_valuerangeprocessor](#) ([Xapian::ValueRangeProcessor](#) \*vrproc)  
*Register a [ValueRangeProcessor](#).*
- std::string [get\\_corrected\\_query\\_string](#) () const  
*Get the spelling-corrected query string.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.45.1 Detailed Description

Build a [Xapian::Query](#) object from a user query string.

### 7.45.2 Member Enumeration Documentation

#### 7.45.2.1 enum [Xapian::QueryParser::feature\\_flag](#)

Enum of feature flags.



**Enumerator:**

**FLAG\_BOOLEAN** Support AND, OR, etc and bracketed subexpressions.

**FLAG\_PHRASE** Support quoted phrases.

**FLAG\_LOVEHATE** Support + and -.

**FLAG\_BOOLEAN\_ANY\_CASE** Support AND, OR, etc even if they aren't in ALLCAPS.

**FLAG\_WILDCARD** Support right truncation (e.g. Xap\*).

Currently you can't use wildcards with boolean filter prefixes, or in a phrase (either an explicitly quoted one, or one implicitly generated by hyphens or other punctuation).

NB: You need to tell the [QueryParser](#) object which database to expand wildcards from by calling `set_database`.

**FLAG\_PURE\_NOT** Allow queries such as 'NOT apples'.

These require the use of a list of all documents in the database which is potentially expensive, so this feature isn't enabled by default.

**FLAG\_PARTIAL** Enable partial matching.

Partial matching causes the parser to treat the query as a "partially entered" search. This will automatically treat the final word as a wildcarded match, unless it is followed by whitespace, to produce more stable results from interactive searches.

Currently FLAG\_PARTIAL doesn't do anything if the final word in the query has a boolean filter prefix, or if it is in a phrase (either an explicitly quoted one, or one implicitly generated by hyphens or other punctuation). It also doesn't do anything if the final word is part of a value range.

NB: You need to tell the [QueryParser](#) object which database to expand wildcards from by calling `set_database`.

**FLAG\_SPELLING\_CORRECTION** Enable spelling correction.

For each word in the query which doesn't exist as a term in the database, [Database::get\\_spelling\\_suggestion\(\)](#) will be called and if a suggestion is returned, a corrected version of the query string will be built up which can be read using [QueryParser::get\\_corrected\\_query\\_string\(\)](#). The query returned is based on the uncorrected query string however - if you want a parsed query based on the corrected query string, you must call [QueryParser::parse\\_query\(\)](#) again.

NB: You must also call [set\\_database\(\)](#) for this to work.

**FLAG\_SYNONYM** Enable synonym operator '~'.

NB: You must also call [set\\_database\(\)](#) for this to work.

**FLAG\_AUTO\_SYNONYMS** Enable automatic use of synonyms for single terms.

NB: You must also call [set\\_database\(\)](#) for this to work.

**FLAG\_AUTO\_MULTIWORD\_SYNONYMS** Enable automatic use of synonyms for single terms and groups of terms.

NB: You must also call [set\\_database\(\)](#) for this to work.

**FLAG\_DEFAULT** The default flags.

Used if you don't explicitly pass any to [parse\\_query\(\)](#).

Added in [Xapian 1.0.11](#).

### 7.45.3 Member Function Documentation

#### 7.45.3.1 `void Xapian::QueryParser::add_boolean_prefix (const std::string &field, const std::string &prefix, bool exclusive)`

Add a boolean term prefix allowing the user to restrict a search with a boolean filter specified in the free text query.

For example:

```
qp.add_boolean_prefix("site", "H");
```

This allows the user to restrict a search with `site:xapian.org` which will be converted to `Hxapian.org` combined with any probabilistic query with [Xapian::Query::OP\\_FILTER](#).

If multiple boolean filters are specified in a query for the same prefix, they will be combined with the [Xapian::Query::OP\\_OR](#) operator. Then, if there are boolean filters for different prefixes, they will be combined with the [Xapian::Query::OP\\_AND](#) operator.

Multiple fields can be mapped to the same prefix (so for example you can make `site:` and `domain:` aliases for each other). Instances of fields with different aliases but the same prefix will still be combined with the OR operator.

For example, if `"site"` and `"domain"` map to `"H"`, but `author` maps to `"A"`, a search for `"site:foo domain:bar author:Fred"` will map to `"(Hfoo OR Hbar) AND Afred"`.

As of 1.0.4, you can call this method multiple times with the same value of `field` to allow a single field to be mapped to multiple prefixes. Multiple terms being generated for such a field, and combined with [Xapian::Query::OP\\_OR](#).

Calling this method with an empty string for `field` will cause a [Xapian::InvalidArgumentError](#).

If you call [add\\_prefix\(\)](#) and [add\\_boolean\\_prefix\(\)](#) for the same value of `field`, a [Xapian::InvalidOperationError](#) exception will be thrown.

In 1.0.3 and earlier, subsequent calls to this method with the same value of `field` had no effect.

#### Parameters:

**field** The user visible field name

**prefix** The term prefix to map this to

**exclusive** If true, each document can have at most one value of the field, so [Xapian](#) should combine multiple values with `OP_OR`. If false, each document can have multiple values of the field, so [Xapian](#) combine them with `OP_AND`, as we would with filters with different prefixes. [default: true]

### 7.45.3.2 void Xapian::QueryParser::add\_prefix (const std::string & *field*, const std::string & *prefix*)

Add a probabilistic term prefix.

For example:

```
qp.add_prefix("author", "A");
```

This allows the user to search for author:Orwell which will be converted to a search for the term "Aorwell".

Multiple fields can be mapped to the same prefix. For example, you can make title: and subject: aliases for each other.

As of 1.0.4, you can call this method multiple times with the same value of *field* to allow a single field to be mapped to multiple prefixes. Multiple terms being generated for such a field, and combined with [Xapian::Query::OP\\_OR](#).

If any prefixes are specified for the empty field name (i.e. you call this method with an empty string as the first parameter) these prefixes will be used for terms without a field specifier. If you do this and also specify the `default_prefix` parameter to [parse\\_query\(\)](#), then the `default_prefix` parameter will override.

If the prefix parameter is empty, then "field:word" will produce the term "word" (and this can be one of several prefixes for a particular field, or for terms without a field specifier).

If you call [add\\_prefix\(\)](#) and [add\\_boolean\\_prefix\(\)](#) for the same value of *field*, a [Xapian::InvalidOperationError](#) exception will be thrown.

In 1.0.3 and earlier, subsequent calls to this method with the same value of *field* had no effect.

#### Parameters:

*field* The user visible field name

*prefix* The term prefix to map this to

### 7.45.3.3 std::string Xapian::QueryParser::get\_corrected\_query\_string () const

Get the spelling-corrected query string.

This will only be set if FLAG\_SPELLING\_CORRECTION is specified when [QueryParser::parse\\_query\(\)](#) was last called.

If there were no corrections, an empty string is returned.

### 7.45.3.4 Query::op Xapian::QueryParser::get\_default\_op () const

Get the current default operator.

#### 7.45.3.5 Query Xapian::QueryParser::parse\_query (const std::string & query\_string, unsigned flags = FLAG\_DEFAULT, const std::string & default\_prefix = std::string())

Parse a query.

##### Parameters:

*query\_string* A free-text query as entered by a user

*flags* Zero or more Query::feature\_flag specifying what features the [Query-Parser](#) should support. Combine multiple values with bitwise-or (|) (default FLAG\_DEFAULT).

*default\_prefix* The default term prefix to use (default none). For example, you can pass "A" when parsing an "Author" field.

##### Exceptions:

*If* the query string can't be parsed, then [Xapian::QueryParserError](#) is thrown. You can get an English error message to report to the user by catching it and calling get\_msg() on the caught exception. The current possible values (in case you want to translate them) are:

- Unknown range operation
- parse error
- Syntax: <expression> AND <expression>
- Syntax: <expression> AND NOT <expression>
- Syntax: <expression> NOT <expression>
- Syntax: <expression> OR <expression>
- Syntax: <expression> XOR <expression>

#### 7.45.3.6 void Xapian::QueryParser::set\_database (const Database & db)

Specify the database being searched.

The database is used for wildcard expansion (FLAG\_WILDCARD and FLAG\_PARTIAL), spelling correction (FLAG\_SPELLING\_CORRECTION), and synonyms (FLAG\_SYNONYM, FLAG\_AUTO\_SYNONYMS, and FLAG\_AUTO\_MULTIWORD\_SYNONYMS).

#### 7.45.3.7 void Xapian::QueryParser::set\_default\_op (Query::op default\_op)

Set the default operator.

This operator is used to combine non-filter query items when no explicit operator is used.

The most useful values for this are OP\_OR (the default) and OP\_AND. OP\_NEAR and OP\_PHRASE can also be useful.

So for example, 'weather forecast' is parsed as if it were 'weather OR forecast' by default.

#### 7.45.3.8 void Xapian::QueryParser::set\_max\_wildcard\_expansion (Xapian::termcount *limit*)

Specify the maximum expansion of a wildcard term.

Note: you must also set FLAG\_WILDCARD for wildcard expansion to happen.

##### Parameters:

*limit* The maximum number of terms each wildcard in the query can expand to, or 0 for no limit (which is the default).

#### 7.45.3.9 void Xapian::QueryParser::set\_stemmer (const Xapian::Stem & *stemmer*)

Set the stemmer.

This sets the stemming algorithm which will be used by the query parser. Note that the stemming algorithm will only be used according to the stemming strategy set by [set\\_stemming\\_strategy\(\)](#), which defaults to STEM\_NONE. Therefore, to use a stemming algorithm, you will also need to call [set\\_stemming\\_strategy\(\)](#) with a value other than STEM\_NONE.

#### 7.45.3.10 void Xapian::QueryParser::set\_stemming\_strategy (stem\_strategy *strategy*)

Set the stemming strategy.

This controls how the query parser will apply the stemming algorithm. The default value is STEM\_NONE. The possible values are:

- STEM\_NONE: Don't perform any stemming.
- STEM\_SOME: Search for stemmed forms of terms except for those which start with a capital letter, or are followed by certain characters (currently: `/@<>=*[{"`), or are used with operators which need positional information. Stemmed terms are prefixed with 'Z'.
- STEM\_ALL: Search for stemmed forms of all words (note: no 'Z' prefix is added).

Note that the stemming algorithm is only applied to words in probabilistic fields - boolean filter terms are never stemmed.

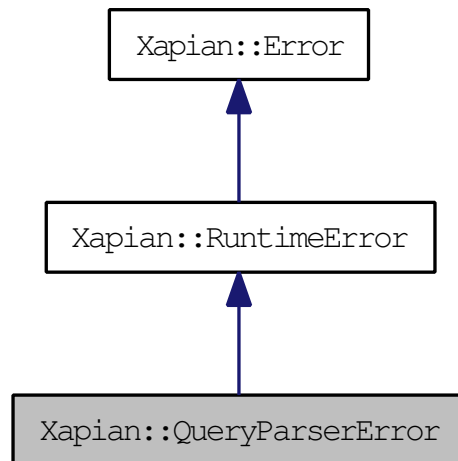
The documentation for this class was generated from the following file:

- [xapian/queryparser.h](#)

## 7.46 Xapian::QueryParserError Class Reference

Indicates a query string can't be parsed.

Inheritance diagram for Xapian::QueryParserError:



### Public Member Functions

- [QueryParserError](#) (const std::string &msg\_, const std::string &context\_  
=std::string(), int errno\_=0)

*General purpose constructor which allows setting errno.*

- [QueryParserError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.46.1 Detailed Description

Indicates a query string can't be parsed.

### 7.46.2 Constructor & Destructor Documentation

- #### 7.46.2.1 Xapian::QueryParserError::QueryParserError (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

#### 7.46.2.2 Xapian::QueryParserError::QueryParserError (const std::string & *msg\_*, int *errno\_*) [inline]

Construct from message and errno value.

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

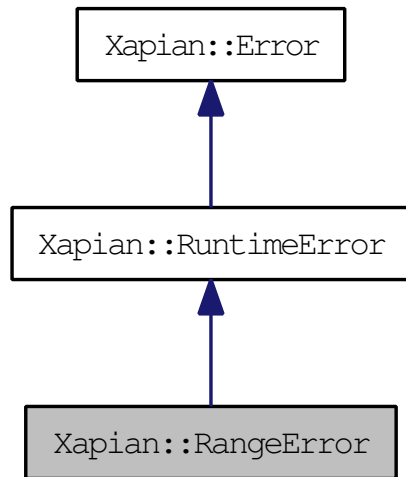
- xapian/[error.h](#)



## 7.47 Xapian::RangeError Class Reference

[RangeError](#) indicates an attempt to access outside the bounds of a container.

Inheritance diagram for Xapian::RangeError:



### Public Member Functions

- [RangeError](#) (const std::string &msg\_, const std::string &context\_=std::string(), int errno\_=0)

*General purpose constructor which allows setting errno.*

- [RangeError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.47.1 Detailed Description

[RangeError](#) indicates an attempt to access outside the bounds of a container.

### 7.47.2 Constructor & Destructor Documentation

- 7.47.2.1** `Xapian::RangeError::RangeError (const std::string & msg_, const std::string & context_ = std::string(), int errno_ = 0) [inline, explicit]`

General purpose constructor which allows setting errno.

#### 7.47.2.2 Xapian::RangeError::RangeError (const std::string & *msg\_*, int *errno\_*) [inline]

Construct from message and errno value.

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

- xapian/[error.h](#)

## 7.48 Xapian::Registry Class Reference

[Registry](#) for user subclasses.

### Public Member Functions

- [Registry](#) (const [Registry](#) &other)  
*Copy constructor.*
- [Registry](#) & operator= (const [Registry](#) &other)  
*Assignment operator.*
- [Registry](#) ()  
*Default constructor.*
- void [register\\_weighting\\_scheme](#) (const [Xapian::Weight](#) &wt)  
*Register a weighting scheme.*
- const [Xapian::Weight](#) \* [get\\_weighting\\_scheme](#) (const std::string &name) const  
*Get the weighting scheme given a name.*
- void [register\\_posting\\_source](#) (const [Xapian::PostingSource](#) &source)  
*Register a user-defined posting source class.*
- const [Xapian::PostingSource](#) \* [get\\_posting\\_source](#) (const std::string &name) const  
*Get a posting source given a name.*
- void [register\\_match\\_spy](#) (const [Xapian::MatchSpy](#) &spy)  
*Register a user-defined match spy class.*
- const [Xapian::MatchSpy](#) \* [get\\_match\\_spy](#) (const std::string &name) const  
*Get a match spy given a name.*

### 7.48.1 Detailed Description

[Registry](#) for user subclasses.

This class provides a way for the remote server to look up user subclasses when unserialising.

## 7.48.2 Constructor & Destructor Documentation

### 7.48.2.1 Xapian::Registry::Registry (const Registry & *other*)

Copy constructor.

The internals are reference counted, so copying is cheap.

### 7.48.2.2 Xapian::Registry::Registry ()

Default constructor.

The registry will contain all standard subclasses of user-subclassable classes.

## 7.48.3 Member Function Documentation

### 7.48.3.1 const Xapian::MatchSpy\* Xapian::Registry::get\_match\_spy (const std::string & *name*) const

Get a match spy given a name.

The returned match spy is owned by the registry object.

Returns NULL if the match spy could not be found.

### 7.48.3.2 const Xapian::PostingSource\* Xapian::Registry::get\_posting\_source (const std::string & *name*) const

Get a posting source given a name.

The returned posting source is owned by the registry object.

Returns NULL if the posting source could not be found.

### 7.48.3.3 const Xapian::Weight\* Xapian::Registry::get\_weighting\_scheme (const std::string & *name*) const

Get the weighting scheme given a name.

The returned weighting scheme is owned by the registry object.

Returns NULL if the weighting scheme could not be found.

### 7.48.3.4 Registry& Xapian::Registry::operator= (const Registry & *other*)

Assignment operator.

The internals are reference counted, so assignment is cheap.

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

- [xapian/registry.h](#)

## 7.49 Xapian::RSet Class Reference

A relevance set (R-Set).

### Public Member Functions

- [RSet](#) (const [RSet](#) &rset)  
*Copy constructor.*
- void [operator=](#) (const [RSet](#) &rset)  
*Assignment operator.*
- [RSet](#) ()  
*Default constructor.*
- [~RSet](#) ()  
*Destructor.*
- [Xapian::doccount size](#) () const  
*The number of documents in this R-Set.*
- bool [empty](#) () const  
*Test if this R-Set is empty.*
- void [add\\_document](#) ([Xapian::docid](#) did)  
*Add a document to the relevance set.*
- void [add\\_document](#) (const [Xapian::MSetIterator](#) &i)  
*Add a document to the relevance set.*
- void [remove\\_document](#) ([Xapian::docid](#) did)  
*Remove a document from the relevance set.*
- void [remove\\_document](#) (const [Xapian::MSetIterator](#) &i)  
*Remove a document from the relevance set.*
- bool [contains](#) ([Xapian::docid](#) did) const  
*Test if a given document in the relevance set.*
- bool [contains](#) (const [Xapian::MSetIterator](#) &i) const  
*Test if a given document in the relevance set.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.49.1 Detailed Description

A relevance set (R-Set).

This is the set of documents which are marked as relevant, for use in modifying the term weights, and in performing query expansion.

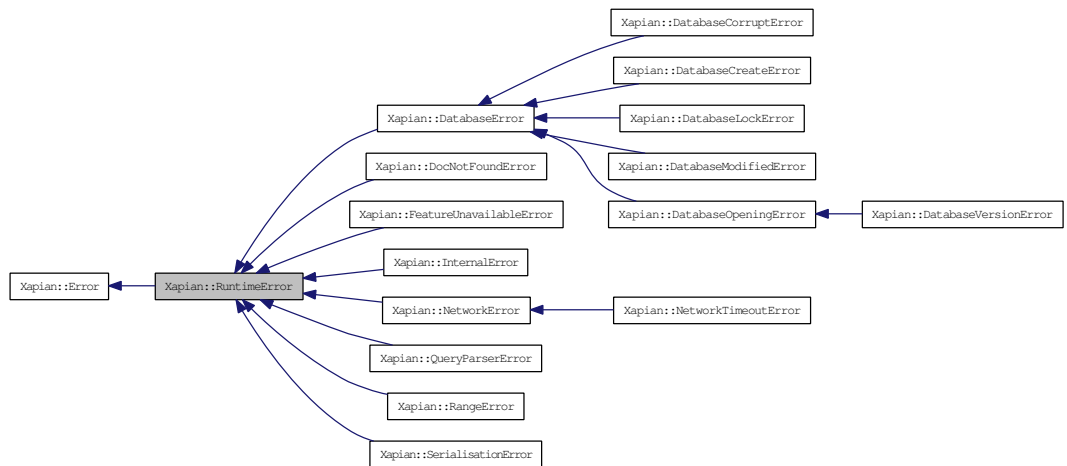
The documentation for this class was generated from the following file:

- [xapian/enquire.h](#)

## 7.50 Xapian::RuntimeError Class Reference

The base class for exceptions indicating errors only detectable at runtime.

Inheritance diagram for Xapian::RuntimeError:



### 7.50.1 Detailed Description

The base class for exceptions indicating errors only detectable at runtime.

A subclass of `RuntimeError` will be thrown if `Xapian` detects an error which is exception derived from `RuntimeError` is thrown when an error is caused by problems with the data or environment rather than a programming mistake.

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

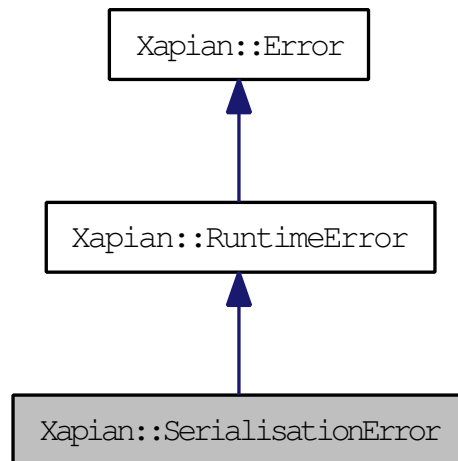
- `xapian/error.h`



## 7.51 Xapian::SerialisationError Class Reference

Indicates an error in the `std::string` serialisation of an object.

Inheritance diagram for Xapian::SerialisationError:



### Public Member Functions

- [SerialisationError](#) (const `std::string` &msg\_, const `std::string` &context\_  
=std::string(), int errno\_=0)

*General purpose constructor which allows setting errno.*

- [SerialisationError](#) (const `std::string` &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.51.1 Detailed Description

Indicates an error in the `std::string` serialisation of an object.

### 7.51.2 Constructor & Destructor Documentation

- #### 7.51.2.1 Xapian::SerialisationError::SerialisationError (const `std::string` &msg\_, const `std::string` &context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

#### 7.51.2.2 Xapian::SerialisationError::SerialisationError (const std::string & *msg\_*, int *errno\_*) [inline]

Construct from message and errno value.

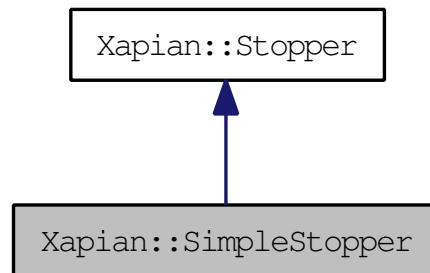
The documentation for this class was generated from the following file:

- xapian/[error.h](#)

## 7.52 Xapian::SimpleStopper Class Reference

Simple implementation of [Stopper](#) class - this will suit most users.

Inheritance diagram for Xapian::SimpleStopper:



### Public Member Functions

- [SimpleStopper](#) ()  
*Default constructor.*
- `template<class Iterator >`  
[SimpleStopper](#) (Iterator begin, Iterator end)  
*Initialise from a pair of iterators.*
- `void` [add](#) (const std::string &word)  
*Add a single stop word.*
- `virtual bool` [operator\(\)](#) (const std::string &term) const  
*Is term a stop-word?*
- `virtual std::string` [get\\_description](#) () const  
*Return a string describing this object.*

### 7.52.1 Detailed Description

Simple implementation of [Stopper](#) class - this will suit most users.

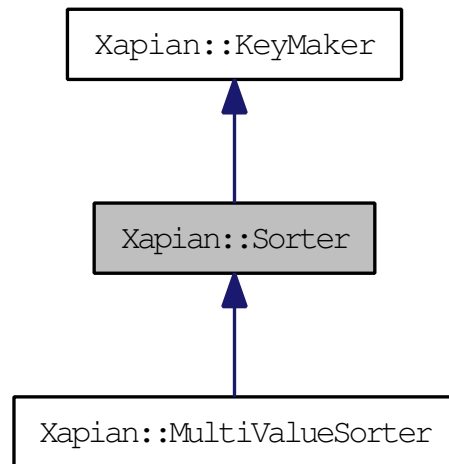
The documentation for this class was generated from the following file:

- [xapian/queryparser.h](#)

## 7.53 Xapian::Sorter Class Reference

Virtual base class for sorter functor.

Inheritance diagram for Xapian::Sorter:



### 7.53.1 Detailed Description

Virtual base class for sorter functor.

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

- [xapian/keymaker.h](#)

## 7.54 Xapian::Stem Class Reference

Class representing a stemming algorithm.

### Public Member Functions

- [Stem](#) (const [Stem](#) &o)  
*Copy constructor.*
- void [operator=](#) (const [Stem](#) &o)  
*Assignment.*
- [Stem](#) ()  
*Construct a [Xapian::Stem](#) object which doesn't change terms.*
- [Stem](#) (const std::string &language)  
*Construct a [Xapian::Stem](#) object for a particular language.*
- [Stem](#) ([StemImplementation](#) \*p)  
*Construct a [Xapian::Stem](#) object with a user-provided stemming algorithm.*
- [~Stem](#) ()  
*Destructor.*
- std::string [operator\(\)](#) (const std::string &word) const  
*[Stem](#) a word.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### Static Public Member Functions

- static std::string [get\\_available\\_languages](#) ()  
*Return a list of available languages.*

#### 7.54.1 Detailed Description

Class representing a stemming algorithm.

## 7.54.2 Constructor & Destructor Documentation

### 7.54.2.1 Xapian::Stem::Stem ()

Construct a [Xapian::Stem](#) object which doesn't change terms.

Equivalent to [Stem](#)("none").

### 7.54.2.2 Xapian::Stem::Stem (const std::string & *language*) [explicit]

Construct a [Xapian::Stem](#) object for a particular language.

#### Parameters:

*language* Either the English name for the language or the two letter ISO639 code.

The following language names are understood (aliases follow the name):

- none - don't stem terms
- danish (da)
- dutch (nl)
- english (en) - Martin Porter's 2002 revision of his stemmer
- english\_lovins (lovins) - Lovin's stemmer
- english\_porter (porter) - Porter's stemmer as described in his 1980 paper
- finnish (fi)
- french (fr)
- german (de)
- german2 - Normalises umlauts and ß
- hungarian (hu)
- italian (it)
- kraaij\_pohlmann - A different Dutch stemmer
- norwegian (nb, nn, no)
- portuguese (pt)
- romanian (ro)
- russian (ru)
- spanish (es)
- swedish (sv)

- `turkish (tr)`

**Exceptions:**

[`Xapian::InvalidArgumentError`](#) is thrown if language isn't recognised.

**7.54.2.3 Xapian::Stem::Stem (StemImplementation \*p) [explicit]**

Construct a [`Xapian::Stem`](#) object with a user-provided stemming algorithm.

You can subclass [`Xapian::StemImplementation`](#) to implement your own stemming algorithm (or to wrap a third-party algorithm) and then wrap your implementation in a [`Xapian::Stem`](#) object to pass to the [`Xapian`](#) API.

The [`StemImplementation`](#) object is reference counted, and so will be automatically deleted by the [`Xapian::Stem`](#) wrapper when no longer required.

**7.54.3 Member Function Documentation****7.54.3.1 static std::string Xapian::Stem::get\_available\_languages ()**  
[static]

Return a list of available languages.

Each stemmer is only included once in the list (not once for each alias). The name included is the English name of the language.

The list is returned as a string, with language names separated by spaces. This is a static method, so a [`Xapian::Stem`](#) object is not required for this operation.

**7.54.3.2 std::string Xapian::Stem::operator() (const std::string & word) const**

[`Stem`](#) a word.

**Parameters:**

*word* a word to stem.

**Returns:**

the stem

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

- [xapian/stem.h](#)

## 7.55 Xapian::StemImplementation Struct Reference

Class representing a stemming algorithm implementation.

### Public Member Functions

- virtual [~StemImplementation](#) ()  
*Virtual destructor.*
- virtual std::string [operator\(\)](#) (const std::string &word)=0  
*[Stem](#) the specified word.*
- virtual std::string [get\\_description](#) () const =0  
*Return a string describing this object.*

#### 7.55.1 Detailed Description

Class representing a stemming algorithm implementation.

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

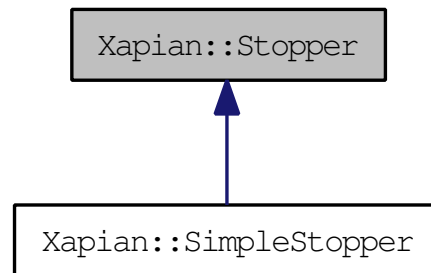
- [xapian/stem.h](#)



## 7.56 Xapian::Stopper Class Reference

Base class for stop-word decision functor.

Inheritance diagram for Xapian::Stopper:



### Public Member Functions

- virtual bool [operator\(\)](#) (const std::string &term) const =0  
*Is term a stop-word?*
- virtual [~Stopper](#) ()  
*Class has virtual methods, so provide a virtual destructor.*
- virtual std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.56.1 Detailed Description

Base class for stop-word decision functor.

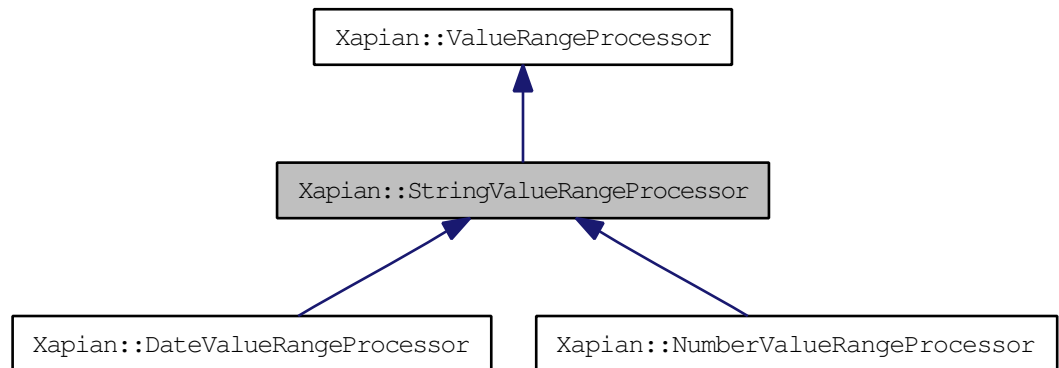
The documentation for this class was generated from the following file:

- [xapian/queryparser.h](#)

## 7.57 Xapian::StringValueRangeProcessor Class Reference

Handle a string range.

Inheritance diagram for Xapian::StringValueRangeProcessor:



### Public Member Functions

- [StringValueRangeProcessor](#) ([Xapian::valueno](#) slot\_)  
*Constructor.*
- [StringValueRangeProcessor](#) ([Xapian::valueno](#) slot\_, const std::string &str\_, bool prefix\_=true)  
*Constructor.*
- [Xapian::valueno operator\(\)](#) (std::string &, std::string &)  
*Check for a valid range of this type.*

### 7.57.1 Detailed Description

Handle a string range.

The end points can be any strings.

### 7.57.2 Constructor & Destructor Documentation

#### 7.57.2.1 Xapian::StringValueRangeProcessor::StringValueRangeProcessor ([Xapian::valueno](#) slot\_) `[inline]`

Constructor.

**Parameters:**

*slot\_* The value number to return from operator().

**7.57.2.2 Xapian::StringValueRangeProcessor::StringValueRangeProcessor**  
(Xapian::valueno *slot\_*, const std::string & *str\_*, bool *prefix\_* = true)  
[inline]

Constructor.

**Parameters:**

*slot\_* The value number to return from operator().

*str\_* A string to look for to recognise values as belonging to this range.

*prefix\_* Flag specifying whether to check for *str\_* as a prefix or a suffix.

**7.57.3 Member Function Documentation**

**7.57.3.1 Xapian::valueno Xapian::StringValueRangeProcessor::operator()**  
(std::string &, std::string &) [virtual]

Check for a valid range of this type.

If no prefix or suffix is specified, then this always returns the value slot specified at construction time.

If a prefix or suffix is specified, this is checked for first, and if it doesn't match, this method returns [Xapian::BAD\\_VALUENO](#).

Implements [Xapian::ValueRangeProcessor](#).

Reimplemented in [Xapian::DateValueRangeProcessor](#), and [Xapian::NumberValueRangeProcessor](#).

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

- [xapian/queryparser.h](#)

## 7.58 Xapian::TermGenerator Class Reference

Parses a piece of text and generate terms.

### Public Types

- enum [flags](#) { [FLAG\\_SPELLING](#) = 128 }  
*Flags to OR together and pass to [TermGenerator::set\\_flags\(\)](#).*

### Public Member Functions

- [TermGenerator](#) (const [TermGenerator](#) &o)  
*Copy constructor.*
- [TermGenerator](#) & [operator=](#) (const [TermGenerator](#) &o)  
*Assignment.*
- [TermGenerator](#) ()  
*Default constructor.*
- [~TermGenerator](#) ()  
*Destructor.*
- void [set\\_stemmer](#) (const [Xapian::Stem](#) &stemmer)  
*Set the [Xapian::Stem](#) object to be used for generating stemmed terms.*
- void [set\\_stopper](#) (const [Xapian::Stopper](#) \*stop=NULL)  
*Set the [Xapian::Stopper](#) object to be used for identifying stopwords.*
- void [set\\_document](#) (const [Xapian::Document](#) &doc)  
*Set the current document.*
- const [Xapian::Document](#) & [get\\_document](#) () const  
*Get the current document.*
- void [set\\_database](#) (const [Xapian::WritableDatabase](#) &db)  
*Set the database to index spelling data to.*
- [flags](#) [set\\_flags](#) ([flags](#) toggle, [flags](#) mask=[flags](#)(0))  
*Set flags.*
- void [index\\_text](#) (const [Xapian::Utf8Iterator](#) &itor, [Xapian::termcount](#) weight=1, const std::string &prefix=std::string())  
*Index some text.*

- void [index\\_text](#) (const std::string &text, [Xapian::termcount](#) weight=1, const std::string &prefix=std::string())  
*Index some text in a std::string.*
- void [index\\_text\\_without\\_positions](#) (const [Xapian::Utf8Iterator](#) &itor, [Xapian::termcount](#) weight=1, const std::string &prefix=std::string())  
*Index some text without positional information.*
- void [index\\_text\\_without\\_positions](#) (const std::string &text, [Xapian::termcount](#) weight=1, const std::string &prefix=std::string())  
*Index some text in a std::string without positional information.*
- void [increase\\_termpos](#) ([Xapian::termcount](#) delta=100)  
*Increase the termpos used by index\_text by delta.*
- [Xapian::termcount](#) [get\\_termpos](#) () const  
*Get the current term position.*
- void [set\\_termpos](#) ([Xapian::termcount](#) termpos)  
*Set the current term position.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.58.1 Detailed Description

Parses a piece of text and generate terms.

This module takes a piece of text and parses it to produce words which are then used to generate suitable terms for indexing. The terms generated are suitable for use with [Query](#) objects produced by the [QueryParser](#) class.

### 7.58.2 Member Enumeration Documentation

#### 7.58.2.1 enum Xapian::TermGenerator::flags

Flags to OR together and pass to [TermGenerator::set\\_flags\(\)](#).

##### Enumerator:

**FLAG\_SPELLING** Index data required for spelling correction.

### 7.58.3 Member Function Documentation

#### 7.58.3.1 `void Xapian::TermGenerator::increase_termpos (Xapian::termcount delta = 100)`

Increase the termpos used by `index_text` by *delta*.

This can be used to prevent phrase searches from spanning two unconnected blocks of text (e.g. the title and body text).

#### 7.58.3.2 `void Xapian::TermGenerator::index_text (const std::string & text, Xapian::termcount weight = 1, const std::string & prefix = std::string()) [inline]`

Index some text in a `std::string`.

##### Parameters:

*weight* The wdf increment (default 1).

*prefix* The term prefix to use (default is no prefix).

#### 7.58.3.3 `void Xapian::TermGenerator::index_text (const Xapian::Utf8Iterator & itor, Xapian::termcount weight = 1, const std::string & prefix = std::string())`

Index some text.

##### Parameters:

*weight* The wdf increment (default 1).

*prefix* The term prefix to use (default is no prefix).

#### 7.58.3.4 `void Xapian::TermGenerator::index_text_without_positions (const std::string & text, Xapian::termcount weight = 1, const std::string & prefix = std::string()) [inline]`

Index some text in a `std::string` without positional information.

Just like `index_text`, but no positional information is generated. This means that the database will be significantly smaller, but that phrase searching and NEAR won't be supported.

#### 7.58.3.5 `void Xapian::TermGenerator::index_text_without_positions (const Xapian::Utf8Iterator & itor, Xapian::termcount weight = 1, const std::string & prefix = std::string())`

Index some text without positional information.

Just like `index_text`, but no positional information is generated. This means that the database will be significantly smaller, but that phrase searching and NEAR won't be supported.

#### 7.58.3.6 `flags` **Xapian::TermGenerator::set\_flags** (*flags toggle*, *flags mask* = `flags (0)`)

Set flags.

The new value of flags is:  $(\text{flags} \& \text{mask}) \wedge \text{toggle}$

To just set the flags, pass the new flags in *toggle* and the default value for *mask*.

##### Parameters:

*toggle* Flags to XOR.

*mask* Flags to AND with first.

##### Returns:

The old flags setting.

#### 7.58.3.7 `void` **Xapian::TermGenerator::set\_stopper** (*const Xapian::Stopper \* stop* = `NULL`)

Set the [Xapian::Stopper](#) object to be used for identifying stopwords.

Stemmed forms of stopwords aren't indexed, but unstemmed forms still are so that searches for phrases including stop words still work.

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

- [xapian/termgenerator.h](#)

## 7.59 Xapian::TermIterator Class Reference

An iterator pointing to items in a list of terms.

### Public Types

- typedef std::input\_iterator\_tag [iterator\\_category](#)  
*Allow use as an STL iterator.*
- typedef std::string [value\\_type](#)  
*Allow use as an STL iterator.*
- typedef [Xapian::termcount\\_diff](#) [difference\\_type](#)  
*Allow use as an STL iterator.*
- typedef std::string \* [pointer](#)  
*Allow use as an STL iterator.*
- typedef std::string & [reference](#)  
*Allow use as an STL iterator.*

### Public Member Functions

- [TermIterator](#) ()  
*Default constructor - for declaring an uninitialised iterator.*
- [~TermIterator](#) ()  
*Destructor.*
- [TermIterator](#) (const [TermIterator](#) &other)  
*Copying is allowed.*
- void [operator=](#) (const [TermIterator](#) &other)  
*Assignment is allowed.*
- std::string [operator\\*](#) () const  
*Return the current term.*
- void [skip\\_to](#) (const std::string &tname)  
*Advance the iterator to the specified term.*
- [Xapian::termcount](#) [get\\_wdf](#) () const  
*Return the wdf of the current term (if meaningful).*
- [Xapian::doccount](#) [get\\_termfreq](#) () const



*Return the term frequency of the current term (if meaningful).*

- [Xapian::termcount positionlist\\_count \(\)](#) const  
*Return length of positionlist for current term.*
- [PositionIterator positionlist\\_begin \(\)](#) const  
*Return [PositionIterator](#) pointing to start of positionlist for current term.*
- [PositionIterator positionlist\\_end \(\)](#) const  
*Return [PositionIterator](#) pointing to end of positionlist for current term.*
- `std::string` [get\\_description \(\)](#) const  
*Return a string describing this object.*

### 7.59.1 Detailed Description

An iterator pointing to items in a list of terms.

### 7.59.2 Constructor & Destructor Documentation

#### 7.59.2.1 Xapian::TermIterator::TermIterator (const TermIterator & other)

Copying is allowed.

The internals are reference counted, so copying is also cheap.

### 7.59.3 Member Function Documentation

#### 7.59.3.1 Xapian::doccount Xapian::TermIterator::get\_termfreq () const

Return the term frequency of the current term (if meaningful).

The term frequency is the number of documents which a term indexes.

#### 7.59.3.2 Xapian::termcount Xapian::TermIterator::get\_wdf () const

Return the wdf of the current term (if meaningful).

The wdf (within document frequency) is the number of occurrences of a term in a particular document.

#### 7.59.3.3 void Xapian::TermIterator::operator= (const TermIterator & other)

Assignment is allowed.

The internals are reference counted, so assignment is also cheap.

**7.59.3.4 void Xapian::TermIterator::skip\_to (const std::string & *tname*)**

Advance the iterator to the specified term.

If the specified term isn't in the list, position ourselves on the first term after it (or `at_end()` if no greater terms are present).

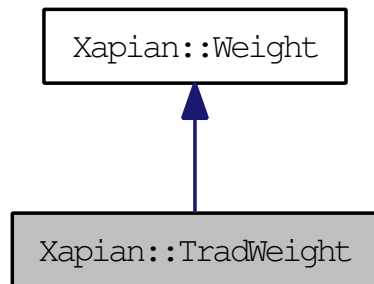
The documentation for this class was generated from the following file:

- `xapian/termiterator.h`

## 7.60 Xapian::TradWeight Class Reference

[Xapian::Weight](#) subclass implementing the traditional probabilistic formula.

Inheritance diagram for Xapian::TradWeight:



### Public Member Functions

- [TradWeight](#) (double k=1.0)  
*Construct a [TradWeight](#).*
- std::string [name](#) () const  
*Return the name of this weighting scheme.*
- std::string [serialise](#) () const  
*Return this object's parameters serialised as a single string.*
- [TradWeight](#) \* [unserialise](#) (const std::string &s) const  
*Unserialise parameters.*
- [Xapian::weight](#) [get\\_sumpart](#) ([Xapian::termcount](#) wdf, [Xapian::termcount](#) doclen) const  
*Calculate the weight contribution for this object's term to a document.*
- [Xapian::weight](#) [get\\_maxpart](#) () const  
*Return an upper bound on what [get\\_sumpart\(\)](#) can return for any document.*
- [Xapian::weight](#) [get\\_sumextra](#) ([Xapian::termcount](#) doclen) const  
*Calculate the term-independent weight component for a document.*
- [Xapian::weight](#) [get\\_maxextra](#) () const  
*Return an upper bound on what [get\\_sumextra\(\)](#) can return for any document.*

### 7.60.1 Detailed Description

[Xapian::Weight](#) subclass implementing the traditional probabilistic formula.

This class implements the "traditional" Probabilistic Weighting scheme, as described by the early papers on Probabilistic Retrieval. BM25 generally gives better results.

TradWeight(k) is equivalent to BM25Weight(k, 0, 0, 1, 0), except that the latter returns weights (k+1) times larger.

### 7.60.2 Constructor & Destructor Documentation

#### 7.60.2.1 `Xapian::TradWeight::TradWeight (double k = 1.0) [inline, explicit]`

Construct a [TradWeight](#).

##### Parameters:

- k* A non-negative parameter controlling how influential within-document-frequency (wdf) and document length are. k=0 means that wdf and document length don't affect the weights. The larger k1 is, the more they do. (default 1)

### 7.60.3 Member Function Documentation

#### 7.60.3.1 `Xapian::weight Xapian::TradWeight::get_maxextra () const [virtual]`

Return an upper bound on what [get\\_sumextra\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implements [Xapian::Weight](#).

#### 7.60.3.2 `Xapian::weight Xapian::TradWeight::get_maxpart () const [virtual]`

Return an upper bound on what [get\\_sumpart\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implements [Xapian::Weight](#).

#### 7.60.3.3 `Xapian::weight Xapian::TradWeight::get_sumextra (Xapian::termcount doclen) const [virtual]`

Calculate the term-independent weight component for a document.

The parameter gives information about the document which may be used in the calculations:

**Parameters:**

*doclen* The document's length (unnormalised).

Implements [Xapian::Weight](#).

#### 7.60.3.4 Xapian::weight Xapian::TradWeight::get\_sumpart (Xapian::termcount *wdf*, Xapian::termcount *doclen*) const [virtual]

Calculate the weight contribution for this object's term to a document.

The parameters give information about the document which may be used in the calculations:

**Parameters:**

*wdf* The within document frequency of the term in the document.

*doclen* The document's length (unnormalised).

Implements [Xapian::Weight](#).

#### 7.60.3.5 std::string Xapian::TradWeight::name () const [virtual]

Return the name of this weighting scheme.

This name is used by the remote backend. It is passed along with the serialised parameters to the remote server so that it knows which class to create.

Return the full namespace-qualified name of your class here - if your class is called FooWeight, return "FooWeight" from this method ([Xapian::BM25Weight](#) returns "Xapian::BM25Weight" here).

If you don't want to support the remote backend, you can use the default implementation which simply returns an empty string.

Reimplemented from [Xapian::Weight](#).

#### 7.60.3.6 std::string Xapian::TradWeight::serialise () const [virtual]

Return this object's parameters serialised as a single string.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::Weight](#).

### 7.60.3.7 TradWeight\* Xapian::TradWeight::unserialise (const std::string & s) const [virtual]

Unserialise parameters.

This method unserialises parameters serialised by the [serialise\(\)](#) method and allocates and returns a new object initialised with them.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented from [Xapian::Weight](#).

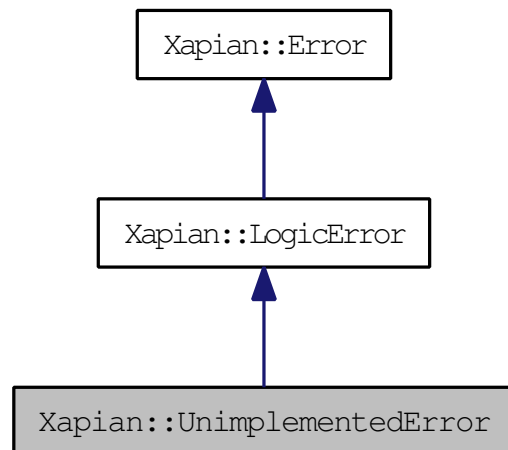
The documentation for this class was generated from the following file:

- [xapian/weight.h](#)

## 7.61 Xapian::UnimplementedError Class Reference

[UnimplementedError](#) indicates an attempt to use an unimplemented feature.

Inheritance diagram for Xapian::UnimplementedError:



### Public Member Functions

- [UnimplementedError](#) (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0)

*General purpose constructor which allows setting errno.*

- [UnimplementedError](#) (const std::string &msg\_, int errno\_)

*Construct from message and errno value.*

### 7.61.1 Detailed Description

[UnimplementedError](#) indicates an attempt to use an unimplemented feature.

### 7.61.2 Constructor & Destructor Documentation

- #### 7.61.2.1 Xapian::UnimplementedError::UnimplementedError (const std::string &msg\_, const std::string &context\_ = std::string(), int errno\_ = 0) [inline, explicit]

General purpose constructor which allows setting errno.

#### 7.61.2.2 Xapian::UnimplementedError::UnimplementedError (const std::string & *msg\_*, int *errno\_*) [inline]

Construct from message and errno value.

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

- [xapian/error.h](#)



## 7.62 Xapian::Utf8Iterator Class Reference

An iterator which returns [Unicode](#) character values from a UTF-8 encoded string.

### Public Types

- typedef std::input\_iterator\_tag [iterator\\_category](#)  
*We implement the semantics of an STL input\_iterator.*
- typedef unsigned [value\\_type](#)  
*We implement the semantics of an STL input\_iterator.*
- typedef size\_t [difference\\_type](#)  
*We implement the semantics of an STL input\_iterator.*
- typedef const unsigned \* [pointer](#)  
*We implement the semantics of an STL input\_iterator.*
- typedef const unsigned & [reference](#)  
*We implement the semantics of an STL input\_iterator.*

### Public Member Functions

- const char \* [raw](#) () const  
*Return the raw const char \* pointer for the current position.*
- size\_t [left](#) () const  
*Return the number of bytes left in the iterator's buffer.*
- void [assign](#) (const char \*p\_, size\_t len)  
*Assign a new string to the iterator.*
- void [assign](#) (const std::string &s)  
*Assign a new string to the iterator.*
- [Utf8Iterator](#) (const char \*p\_)  
*Create an iterator given a pointer to a null terminated string.*
- [Utf8Iterator](#) (const char \*p\_, size\_t len)  
*Create an iterator given a pointer and a length.*
- [Utf8Iterator](#) (const std::string &s)  
*Create an iterator given a string.*
- [Utf8Iterator](#) ()

*Create an iterator which is at the end of its iteration.*

- unsigned `operator*` () const  
*Get the current [Unicode](#) character value pointed to by the iterator.*
- `Utf8Iterator operator++` (int)  
*Move forward to the next [Unicode](#) character.*
- `Utf8Iterator & operator++` ()  
*Move forward to the next [Unicode](#) character.*
- bool `operator==` (const `Utf8Iterator` &other) const  
*Test two `Utf8Iterators` for equality.*
- bool `operator!=` (const `Utf8Iterator` &other) const  
*Test two `Utf8Iterators` for inequality.*

### 7.62.1 Detailed Description

An iterator which returns [Unicode](#) character values from a UTF-8 encoded string.

### 7.62.2 Constructor & Destructor Documentation

#### 7.62.2.1 `Xapian::Utf8Iterator::Utf8Iterator (const char * p_)` `[explicit]`

Create an iterator given a pointer to a null terminated string.

The iterator will return characters from the start of the string when next called. The string is not copied into the iterator, so it must remain valid while the iteration is in progress.

#### Parameters:

*p\_* A pointer to the start of the null terminated string to read.

#### 7.62.2.2 `Xapian::Utf8Iterator::Utf8Iterator (const char * p_, size_t len)` `[inline]`

Create an iterator given a pointer and a length.

The iterator will return characters from the start of the string when next called. The string is not copied into the iterator, so it must remain valid while the iteration is in progress.

#### Parameters:

*p\_* A pointer to the start of the string to read.

*len* The length of the string to read.

#### 7.62.2.3 Xapian::Utf8Iterator::Utf8Iterator (const std::string & s) [inline]

Create an iterator given a string.

The iterator will return characters from the start of the string when next called. The string is not copied into the iterator, so it must remain valid while the iteration is in progress.

##### Parameters:

*s* The string to read. Must not be modified while the iteration is in progress.

#### 7.62.2.4 Xapian::Utf8Iterator::Utf8Iterator () [inline]

Create an iterator which is at the end of its iteration.

This can be compared to another iterator to check if the other iterator has reached its end.

### 7.62.3 Member Function Documentation

#### 7.62.3.1 void Xapian::Utf8Iterator::assign (const std::string & s) [inline]

Assign a new string to the iterator.

The iterator will forget the string it was iterating through, and return characters from the start of the new string when next called. The string is not copied into the iterator, so it must remain valid while the iteration is in progress.

##### Parameters:

*s* The string to read. Must not be modified while the iteration is in progress.

References assign().

Referenced by assign().

#### 7.62.3.2 void Xapian::Utf8Iterator::assign (const char \* p\_, size\_t len) [inline]

Assign a new string to the iterator.

The iterator will forget the string it was iterating through, and return characters from the start of the new string when next called. The string is not copied into the iterator, so it must remain valid while the iteration is in progress.

**Parameters:**

*p\_* A pointer to the start of the string to read.

*len* The length of the string to read.

**7.62.3.3** `size_t Xapian::Utf8Iterator::left () const` `[inline]`

Return the number of bytes left in the iterator's buffer.

**7.62.3.4** `bool Xapian::Utf8Iterator::operator!= (const Utf8Iterator & other)`  
`const` `[inline]`

Test two Utf8Iterators for inequality.

**Returns:**

true iff the iterators do not point to the same position.

**7.62.3.5** `unsigned Xapian::Utf8Iterator::operator* () const`

Get the current [Unicode](#) character value pointed to by the iterator.

Returns unsigned(-1) if the iterator has reached the end of its buffer.

**7.62.3.6** `Utf8Iterator& Xapian::Utf8Iterator::operator++ ()` `[inline]`

Move forward to the next [Unicode](#) character.

**Returns:**

A reference to this object.

**7.62.3.7** `Utf8Iterator Xapian::Utf8Iterator::operator++ (int)` `[inline]`

Move forward to the next [Unicode](#) character.

**Returns:**

An iterator pointing to the position before the move.

**7.62.3.8** `bool Xapian::Utf8Iterator::operator== (const Utf8Iterator & other)`  
`const` `[inline]`

Test two Utf8Iterators for equality.

**Returns:**

true iff the iterators point to the same position.

**7.62.3.9** `const char* Xapian::Utf8Iterator::raw () const` `[inline]`

Return the raw const char \* pointer for the current position.

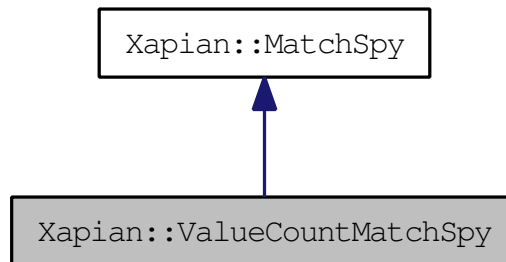
The documentation for this class was generated from the following file:

- [xapian/unicode.h](#)

## 7.63 Xapian::ValueCountMatchSpy Class Reference

Class for counting the frequencies of values in the matching documents.

Inheritance diagram for Xapian::ValueCountMatchSpy:



### Public Member Functions

- [ValueCountMatchSpy](#) ()  
*Construct an empty [ValueCountMatchSpy](#).*
- [ValueCountMatchSpy](#) (Xapian::valueno slot\_)  
*Construct a [MatchSpy](#) which counts the values in a particular slot.*
- size\_t [get\\_total](#) () const  
*Return the total number of documents tallied.*
- [TermIterator](#) [values\\_begin](#) () const  
*Get an iterator over the values seen in the slot.*
- [TermIterator](#) [values\\_end](#) () const  
*End iterator corresponding to [values\\_begin](#)().*
- [TermIterator](#) [top\\_values\\_begin](#) (size\_t maxvalues) const  
*Get an iterator over the most frequent values seen in the slot.*
- [TermIterator](#) [top\\_values\\_end](#) (size\_t) const  
*End iterator corresponding to [top\\_values\\_begin](#)().*
- void [operator\(\)](#) (const Xapian::Document &doc, Xapian::weight wt)  
*Implementation of virtual operator().*
- virtual [MatchSpy](#) \* [clone](#) () const  
*Clone the match spy.*
- virtual std::string [name](#) () const

*Return the name of this match spy.*

- virtual std::string [serialise](#) () const

*Return this object's parameters serialised as a single string.*

- virtual [MatchSpy](#) \* [unserialise](#) (const std::string &s, const [Registry](#) &context) const

*Unserialise parameters.*

- virtual std::string [serialise\\_results](#) () const

*Serialise the results of this match spy.*

- virtual void [merge\\_results](#) (const std::string &s)

*Unserialise some results, and merge them into this matchspy.*

- virtual std::string [get\\_description](#) () const

*Return a string describing this object.*

### 7.63.1 Detailed Description

Class for counting the frequencies of values in the matching documents.

### 7.63.2 Member Function Documentation

#### 7.63.2.1 virtual [MatchSpy](#)\* [Xapian::ValueCountMatchSpy::clone](#) () const [virtual]

Clone the match spy.

The clone should inherit the configuration of the parent, but need not inherit the state. ie, the clone does not need to be passed information about the results seen by the parent.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented from [Xapian::MatchSpy](#).

#### 7.63.2.2 virtual std::string [Xapian::ValueCountMatchSpy::get\\_description](#) () const [virtual]

Return a string describing this object.

This default implementation returns a generic answer, to avoid forcing those deriving their own [MatchSpy](#) subclasses from having to implement this (they may not care what [get\\_description\(\)](#) gives for their subclass).

Reimplemented from [Xapian::MatchSpy](#).

#### 7.63.2.3 `size_t Xapian::ValueCountMatchSpy::get_total () const` [inline]

Return the total number of documents tallied.

#### 7.63.2.4 `virtual void Xapian::ValueCountMatchSpy::merge_results (const std::string & s)` [virtual]

Unserialise some results, and merge them into this matchspy.

The order in which results are merged should not be significant, since this order is not specified (and will vary depending on the speed of the search in each sub-database).

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::MatchSpy](#).

#### 7.63.2.5 `virtual std::string Xapian::ValueCountMatchSpy::name () const` [virtual]

Return the name of this match spy.

This name is used by the remote backend. It is passed with the serialised parameters to the remote server so that it knows which class to create.

Return the full namespace-qualified name of your class here - if your class is called `MyApp::FooMatchSpy`, return `"MyApp::FooMatchSpy"` from this method.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::MatchSpy](#).

#### 7.63.2.6 `void Xapian::ValueCountMatchSpy::operator() (const Xapian::Document & doc, Xapian::weight wt)` [virtual]

Implementation of virtual `operator()`.

This implementation tallies values for a matching document.

Implements [Xapian::MatchSpy](#).

#### 7.63.2.7 `virtual std::string Xapian::ValueCountMatchSpy::serialise () const` [virtual]

Return this object's parameters serialised as a single string.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).



Reimplemented from [Xapian::MatchSpy](#).

#### 7.63.2.8 **virtual std::string Xapian::ValueCountMatchSpy::serialise\_results () const** [virtual]

Serialise the results of this match spy.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::MatchSpy](#).

#### 7.63.2.9 **TermIterator Xapian::ValueCountMatchSpy::top\_values\_begin (size\_t maxvalues) const**

Get an iterator over the most frequent values seen in the slot.

Items will be returned in descending order of frequency. Values with the same frequency will be returned in ascending alphabetical order.

During the iteration, the frequency of the current value can be obtained with the `get_termfreq()` method on the iterator.

##### Parameters:

*maxvalues* The maximum number of values to return.

#### 7.63.2.10 **virtual MatchSpy\* Xapian::ValueCountMatchSpy::unserialise (const std::string & s, const Registry & context) const** [virtual]

Unserialise parameters.

This method unserialises parameters serialised by the [serialise\(\)](#) method and allocates and returns a new object initialised with them.

If you don't want to support the remote backend in your match spy, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented from [Xapian::MatchSpy](#).

#### 7.63.2.11 **TermIterator Xapian::ValueCountMatchSpy::values\_begin () const**

Get an iterator over the values seen in the slot.

Items will be returned in ascending alphabetical order.

During the iteration, the frequency of the current value can be obtained with the `get_termfreq()` method on the iterator.

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

- [xapian/matchspy.h](#)

## 7.64 Xapian::ValueIterator Class Reference

Class for iterating over document values.

### Public Member Functions

- [ValueIterator](#) (const [ValueIterator](#) &o)  
*Copy constructor.*
- [ValueIterator](#) & [operator=](#) (const [ValueIterator](#) &o)  
*Assignment.*
- [ValueIterator](#) ()  
*Default constructor.*
- [~ValueIterator](#) ()  
*Destructor.*
- std::string [operator\\*](#) () const  
*Return the value at the current position.*
- [ValueIterator](#) & [operator++](#) ()  
*Advance the iterator to the next position.*
- DerefWrapper\_< std::string > [operator++](#) (int)  
*Advance the iterator to the next position (postfix version).*
- [Xapian::docid](#) [get\\_docid](#) () const  
*Return the docid at the current position.*
- [Xapian::valueno](#) [get\\_valueno](#) () const  
*Return the value slot number for the current position.*
- void [skip\\_to](#) ([Xapian::docid](#) docid\_or\_slot)  
*Advance the iterator to document id or value slot docid\_or\_slot.*
- bool [check](#) ([Xapian::docid](#) docid)  
*Check if the specified docid occurs.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.64.1 Detailed Description

Class for iterating over document values.

## 7.64.2 Constructor & Destructor Documentation

### 7.64.2.1 Xapian::ValueIterator::ValueIterator ()

Default constructor.

Creates an uninitialised iterator, which can't be used before being assigned to, but is sometimes syntactically convenient.

## 7.64.3 Member Function Documentation

### 7.64.3.1 bool Xapian::ValueIterator::check (Xapian::docid *docid*)

Check if the specified docid occurs.

The caller is required to ensure that the specified document id *did* actually exists in the database.

This method acts like [skip\\_to\(\)](#) if that can be done at little extra cost, in which case it then returns true. This is how brass and chert databases behave because they store values in streams which allow for an efficient implementation of [skip\\_to\(\)](#).

Otherwise it simply checks if a particular docid is present. If it is, it returns true. If it isn't, it returns false, and leaves the position unspecified (and hence the result of calling methods which depends on the current position, such as [get\\_docid\(\)](#), are also unspecified). In this state, [next\(\)](#) will advance to the first matching position after document *did*, and [skip\\_to\(\)](#) will act as it would if the position was the first matching position after document *did*.

Currently the inmemory, flint, and remote backends behave in the latter way because they don't support streamed values and so [skip\\_to\(\)](#) must check each document it skips over which is significantly slower.

### 7.64.3.2 Xapian::docid Xapian::ValueIterator::get\_docid () const

Return the docid at the current position.

If we're iterating over values of a document, this method will throw [Xapian::InvalidOperationError](#).

### 7.64.3.3 Xapian::valueno Xapian::ValueIterator::get\_valueno () const

Return the value slot number for the current position.

If the iterator is over all values in a slot, this returns that slot's number. If the iterator is over the values in a particular document, it returns the number of each slot in turn.

### 7.64.3.4 void Xapian::ValueIterator::skip\_to (Xapian::docid *docid\_or\_slot*)

Advance the iterator to document id or value slot *docid\_or\_slot*.

If this iterator is over values in a document, then this method advances the iterator to value slot *docid\_or\_slot*, or the first slot after it if there is no value in slot *slot*.

If this iterator is over values in a particular slot, then this method advances the iterator to document id *docid\_or\_slot*, or the first document id after it if there is no value in the slot we're iterating over for document *docid\_or\_slot*.

Note: The "two-faced" nature of this method is due to how C++ overloading works. [Xapian::docid](#) and [Xapian::valueno](#) are both typedefs for the same unsigned integer type, so overloading can't distinguish them.

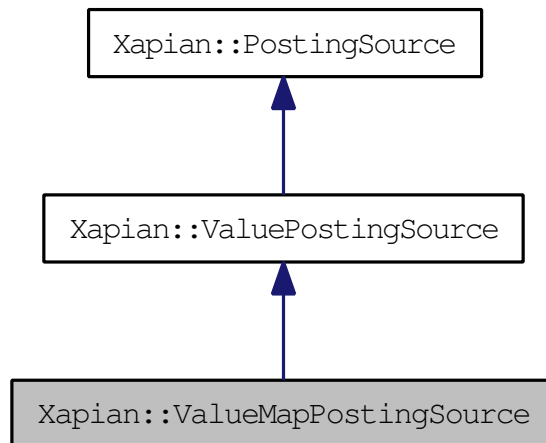
The documentation for this class was generated from the following file:

- [xapian/valueiterator.h](#)

## 7.65 Xapian::ValueMapPostingSource Class Reference

A posting source which looks up weights in a map using values as the key.

Inheritance diagram for Xapian::ValueMapPostingSource:



### Public Member Functions

- [ValueMapPostingSource](#) ([Xapian::valueno](#) slot\_)  
*Construct a [ValueWeightPostingSource](#).*
- void [add\\_mapping](#) (const std::string &key, double [weight](#))  
*Add a mapping.*
- void [clear\\_mappings](#) ()  
*Clear all mappings.*
- void [set\\_default\\_weight](#) (double wt)  
*Set a default weight for document values not in the map.*
- [Xapian::weight](#) [get\\_weight](#) () const  
*Return the weight contribution for the current document.*
- [ValueMapPostingSource](#) \* [clone](#) () const  
*Clone the posting source.*
- std::string [name](#) () const  
*Name of the posting source class.*
- std::string [serialise](#) () const  
*Serialise object parameters into a string.*

- [ValueMapPostingSource](#) \* [unserialise](#) (const std::string &s) const  
*Create object given string serialisation returned by [serialise\(\)](#).*
- void [init](#) (const [Database](#) &db\_)  
*Set this [PostingSource](#) to the start of the list of postings.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.65.1 Detailed Description

A posting source which looks up weights in a map using values as the key.

This allows will return entries for all documents in the given database which have a value in the slot specified. The values will be mapped to the corresponding weight in the weight map. If there is no mapping for a particular value, the default weight will be returned (which itself defaults to 0.0).

### 7.65.2 Constructor & Destructor Documentation

#### 7.65.2.1 Xapian::ValueMapPostingSource::ValueMapPostingSource (Xapian::valueno slot\_)

Construct a [ValueWeightPostingSource](#).

##### Parameters:

*slot\_* The value slot to read values from.

### 7.65.3 Member Function Documentation

#### 7.65.3.1 void Xapian::ValueMapPostingSource::add\_mapping (const std::string &key, double weight)

Add a mapping.

##### Parameters:

*key* The key looked up from the value slot.

*weight* The weight to give this key.

#### 7.65.3.2 void Xapian::ValueMapPostingSource::clear\_mappings ()

Clear all mappings.

### 7.65.3.3 `ValueMapPostingSource* Xapian::ValueMapPostingSource::clone () const` [virtual]

Clone the posting source.

The clone should inherit the configuration of the parent, but need not inherit the state. ie, the clone does not need to be in the same iteration position as the original: the matcher will always call `init()` on the clone before attempting to move the iterator, or read the information about the current position of the iterator.

This may return NULL to indicate that cloning is not supported. In this case, the `PostingSource` may only be used with a single-database search.

The default implementation returns NULL.

Note that the returned object will be deallocated by `Xapian` after use with "delete". It must therefore have been allocated with "new".

Reimplemented from `Xapian::PostingSource`.

### 7.65.3.4 `std::string Xapian::ValueMapPostingSource::get_description () const` [virtual]

Return a string describing this object.

This default implementation returns a generic answer. This default is provided to avoid forcing those deriving their own `PostingSource` subclass from having to implement this (they may not care what `get_description()` gives for their subclass).

Reimplemented from `Xapian::PostingSource`.

### 7.65.3.5 `Xapian::weight Xapian::ValueMapPostingSource::get_weight () const` [virtual]

Return the weight contribution for the current document.

This default implementation always returns 0, for convenience when implementing "weight-less" `PostingSource` subclasses.

This method may assume that it will only be called when there is a "current document". In detail: `Xapian` will always call `init()` on a `PostingSource` before calling this for the first time. It will also only call this if the `PostingSource` reports that it is pointing to a valid document (ie, it will not call it before calling at least one of `next()`, `skip_to()` or `check()`, and will ensure that the `PostingSource` is not at the end by calling `at_end()`).

Reimplemented from `Xapian::PostingSource`.

### 7.65.3.6 `void Xapian::ValueMapPostingSource::init (const Database & db)` [virtual]

Set this `PostingSource` to the start of the list of postings.

This is called automatically by the matcher prior to each query being processed.



If a [PostingSource](#) is used for multiple searches, [init\(\)](#) will therefore be called multiple times, and must handle this by using the database passed in the most recent call.

**Parameters:**

*db* The database which the [PostingSource](#) should iterate through.

Note: the database supplied to this method must not be modified: in particular, the [reopen\(\)](#) method should not be called on it.

Note: in the case of a multi-database search, a separate [PostingSource](#) will be used for each database (the separate PostingSources will be obtained using [clone\(\)](#)), and each [PostingSource](#) will be passed one of the sub-databases as the *db* parameter here. The *db* parameter will therefore always refer to a single database. All docids passed to, or returned from, the [PostingSource](#) refer to docids in that single database, rather than in the multi-database.

Reimplemented from [Xapian::ValuePostingSource](#).

### 7.65.3.7 `std::string Xapian::ValueMapPostingSource::name () const` [virtual]

Name of the posting source class.

This is used when serialising and unserialising posting sources; for example, for performing remote searches.

If the subclass is in a C++ namespace, the namespace should be included in the name, using ":" as a separator. For example, for a [PostingSource](#) subclass called "FooPostingSource" in the "Xapian" namespace the result of this call should be "Xapian::FooPostingSource".

This should only be implemented if [serialise\(\)](#) and [unserialise\(\)](#) are also implemented. The default implementation returns an empty string.

If this returns an empty string, [Xapian](#) will assume that [serialise\(\)](#) and [unserialise\(\)](#) are not implemented.

Reimplemented from [Xapian::PostingSource](#).

### 7.65.3.8 `std::string Xapian::ValueMapPostingSource::serialise () const` [virtual]

Serialise object parameters into a string.

The serialised parameters should represent the configuration of the posting source, but need not (indeed, should not) represent the current iteration state.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::PostingSource](#).

### 7.65.3.9 void Xapian::ValueMapPostingSource::set\_default\_weight (double *wt*)

Set a default weight for document values not in the map.

### 7.65.3.10 ValueMapPostingSource\* Xapian::ValueMapPostingSource::unserialise (const std::string & *s*) const [virtual]

Create object given string serialisation returned by [serialise\(\)](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

#### Parameters:

*s* A serialised instance of this [PostingSource](#) subclass.

Reimplemented from [Xapian::PostingSource](#).

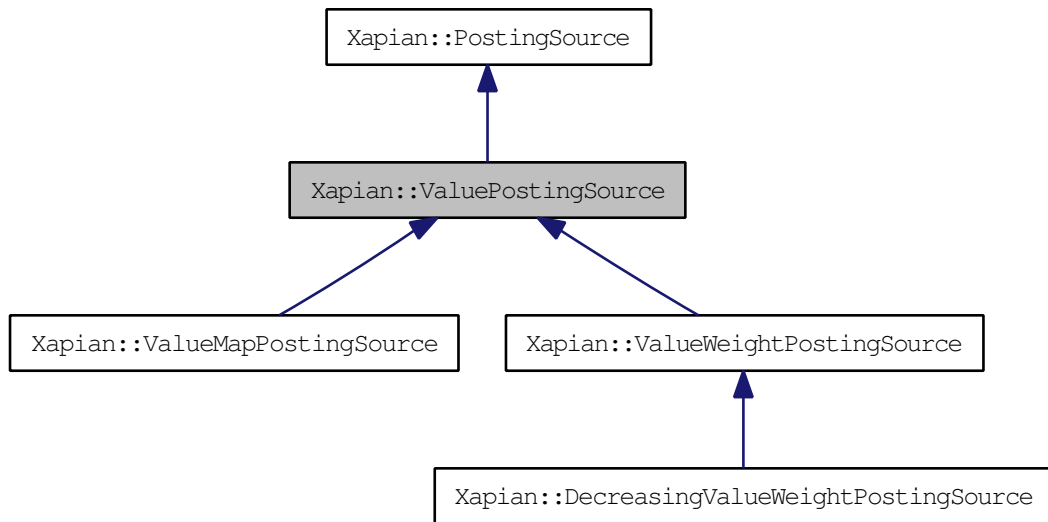
The documentation for this class was generated from the following file:

- [xapian/postingsource.h](#)

## 7.66 Xapian::ValuePostingSource Class Reference

A posting source which generates weights from a value slot.

Inheritance diagram for Xapian::ValuePostingSource:



### Public Member Functions

- [ValuePostingSource](#) ([Xapian::valueno](#) slot\_)  
*Construct a [ValuePostingSource](#).*
- [Xapian::doccount get\\_termfreq\\_min](#) () const  
*A lower bound on the number of documents this object can return.*
- [Xapian::doccount get\\_termfreq\\_est](#) () const  
*An estimate of the number of documents this object can return.*
- [Xapian::doccount get\\_termfreq\\_max](#) () const  
*An upper bound on the number of documents this object can return.*
- void [next](#) ([Xapian::weight](#) min\_wt)  
*Advance the current position to the next matching document.*
- void [skip\\_to](#) ([Xapian::docid](#) min\_docid, [Xapian::weight](#) min\_wt)  
*Advance to the specified docid.*
- bool [check](#) ([Xapian::docid](#) min\_docid, [Xapian::weight](#) min\_wt)  
*Check if the specified docid occurs.*

- `bool at_end () const`  
*Return true if the current position is past the last entry in this list.*
- `Xapian::docid get_docid () const`  
*Return the current docid.*
- `void init (const Database &db_)`  
*Set this [PostingSource](#) to the start of the list of postings.*

## Protected Attributes

- `Xapian::Database db`  
*The database we're reading values from.*
- `Xapian::valueno slot`  
*The slot we're reading values from.*
- `Xapian::ValueIterator value_it`  
*Value stream iterator.*
- `bool started`  
*Flag indicating if we've started (true if we have).*
- `Xapian::doccount termfreq_min`  
*A lower bound on the term frequency.*
- `Xapian::doccount termfreq_est`  
*An estimate of the term frequency.*
- `Xapian::doccount termfreq_max`  
*An upper bound on the term frequency.*

### 7.66.1 Detailed Description

A posting source which generates weights from a value slot.

This is a base class for classes which generate weights using values stored in the specified slot. For example, [ValueWeightPostingSource](#) uses `sortable_unserialise` to convert values directly to weights.

The upper bound on the weight returned is set to `DBL_MAX`. Subclasses should call [set\\_maxweight\(\)](#) in their `init()` methods after calling [ValuePostingSource::init\(\)](#) if they know a tighter bound on the weight.

## 7.66.2 Constructor & Destructor Documentation

### 7.66.2.1 Xapian::ValuePostingSource::ValuePostingSource (Xapian::valueno *slot\_*)

Construct a [ValuePostingSource](#).

#### Parameters:

*slot\_* The value slot to read values from.

## 7.66.3 Member Function Documentation

### 7.66.3.1 bool Xapian::ValuePostingSource::at\_end () const [virtual]

Return true if the current position is past the last entry in this list.

At least one of [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#) will be called before this method is first called.

Implements [Xapian::PostingSource](#).

### 7.66.3.2 bool Xapian::ValuePostingSource::check (Xapian::docid *did*, Xapian::weight *min\_wt*) [virtual]

Check if the specified docid occurs.

The caller is required to ensure that the specified document id *did* actually exists in the database. If it does, it must move to that document id, and return true. If it does not, it may either:

- return true, having moved to a definite position (including "at\_end"), which must be the same position as [skip\\_to\(\)](#) would have moved to.

or

- return false, having moved to an "indeterminate" position, such that a subsequent call to [next\(\)](#) or [skip\\_to\(\)](#) will move to the next matching position after *did*.

Generally, this method should act like [skip\\_to\(\)](#) and return true if that can be done at little extra cost.

Otherwise it should simply check if a particular docid is present, returning true if it is, and false if it isn't.

The default implementation calls [skip\\_to\(\)](#) and always returns true.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

#### 7.66.3.3 **Xapian::docid Xapian::ValuePostingSource::get\_docid () const** [virtual]

Return the current docid.

This method may assume that it will only be called when there is a "current document". See [get\\_weight\(\)](#) for details.

Note: in the case of a multi-database search, the returned docid should be in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

Implements [Xapian::PostingSource](#).

#### 7.66.3.4 **Xapian::doccount Xapian::ValuePostingSource::get\_termfreq\_est () const** [virtual]

An estimate of the number of documents this object can return.

It must always be true that:

[get\\_termfreq\\_min\(\)](#) <= [get\\_termfreq\\_est\(\)](#) <= [get\\_termfreq\\_max\(\)](#)

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implements [Xapian::PostingSource](#).

#### 7.66.3.5 **Xapian::doccount Xapian::ValuePostingSource::get\_termfreq\_max () const** [virtual]

An upper bound on the number of documents this object can return.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implements [Xapian::PostingSource](#).

#### 7.66.3.6 **Xapian::doccount Xapian::ValuePostingSource::get\_termfreq\_min () const** [virtual]

A lower bound on the number of documents this object can return.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Implements [Xapian::PostingSource](#).

#### 7.66.3.7 **void Xapian::ValuePostingSource::init (const Database & db)** [virtual]

Set this [PostingSource](#) to the start of the list of postings.

This is called automatically by the matcher prior to each query being processed.

If a [PostingSource](#) is used for multiple searches, [init\(\)](#) will therefore be called multiple times, and must handle this by using the database passed in the most recent call.

#### Parameters:

*db* The database which the [PostingSource](#) should iterate through.

Note: the database supplied to this method must not be modified: in particular, the [reopen\(\)](#) method should not be called on it.

Note: in the case of a multi-database search, a separate [PostingSource](#) will be used for each database (the separate PostingSources will be obtained using [clone\(\)](#)), and each [PostingSource](#) will be passed one of the sub-databases as the *db* parameter here. The *db* parameter will therefore always refer to a single database. All docids passed to, or returned from, the [PostingSource](#) refer to docids in that single database, rather than in the multi-database.

Implements [Xapian::PostingSource](#).

Reimplemented in [Xapian::ValueWeightPostingSource](#),  
[Xapian::DecreasingValueWeightPostingSource](#), and  
[Xapian::ValueMapPostingSource](#).

#### 7.66.3.8 void Xapian::ValuePostingSource::next (Xapian::weight *min\_wt*) [virtual]

Advance the current position to the next matching document.

The [PostingSource](#) starts before the first entry in the list, so [next\(\)](#) must be called before any methods which need the context of the current position.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

#### Parameters:

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Implements [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

#### 7.66.3.9 void Xapian::ValuePostingSource::skip\_to (Xapian::docid *did*, Xapian::weight *min\_wt*) [virtual]

Advance to the specified docid.

If the specified docid isn't in the list, position ourselves on the first document after it (or [at\\_end\(\)](#) if no greater docids are present).

If the current position is already the specified docid, this method will leave the position unmodified.

If the specified docid is earlier than the current position, the behaviour is unspecified. A sensible behaviour would be to leave the current position unmodified, but it is also reasonable to move to the specified docid.

The default implementation calls [next\(\)](#) repeatedly, which works but [skip\\_to\(\)](#) can often be implemented much more efficiently.

[Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time.

Note: in the case of a multi-database search, the docid specified is the docid in the single subdatabase relevant to this posting source. See the [init\(\)](#) method for details.

#### Parameters:

*min\_wt* The minimum weight contribution that is needed (this is just a hint which subclasses may ignore).

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

## 7.66.4 Member Data Documentation

### 7.66.4.1 [Xapian::doccount Xapian::ValuePostingSource::termfreq\\_est](#) [protected]

An estimate of the term frequency.

Subclasses should set this if they are overriding the [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#) methods.

### 7.66.4.2 [Xapian::doccount Xapian::ValuePostingSource::termfreq\\_max](#) [protected]

An upper bound on the term frequency.

Subclasses should set this if they are overriding the [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#) methods.

### 7.66.4.3 [Xapian::doccount Xapian::ValuePostingSource::termfreq\\_min](#) [protected]

A lower bound on the term frequency.

Subclasses should set this if they are overriding the [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#) methods to return fewer documents.

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

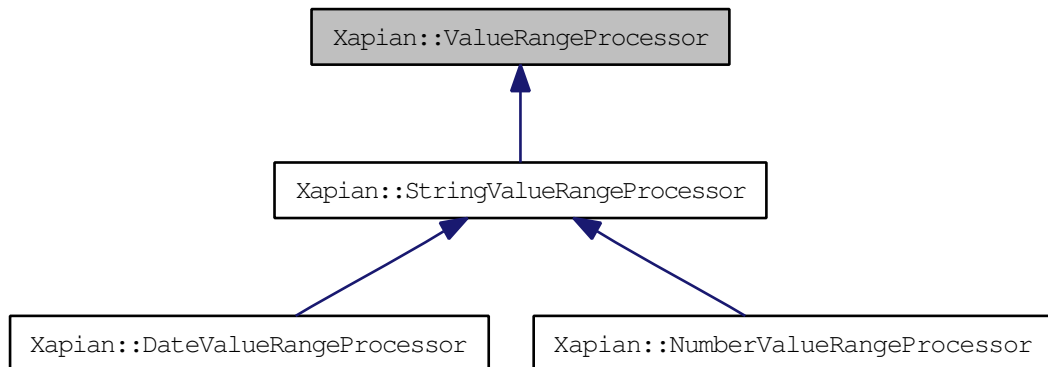
- [xapian/postingsource.h](#)



## 7.67 Xapian::ValueRangeProcessor Struct Reference

Base class for value range processors.

Inheritance diagram for Xapian::ValueRangeProcessor:



### Public Member Functions

- virtual [~ValueRangeProcessor](#) ()  
*Destructor.*
- virtual [Xapian::valueno operator\(\)](#) (std::string &begin, std::string &end)=0  
*Check for a valid range of this type.*

### 7.67.1 Detailed Description

Base class for value range processors.

### 7.67.2 Member Function Documentation

#### 7.67.2.1 virtual Xapian::valueno Xapian::ValueRangeProcessor::operator() (std::string &begin, std::string &end) [pure virtual]

Check for a valid range of this type.

If this [ValueRangeProcessor](#) recognises BEGIN..END it returns the value number to range filter on. Otherwise it returns [Xapian::BAD\\_VALUENO](#).

Implemented in [Xapian::StringValueRangeProcessor](#), [Xapian::DateValueRangeProcessor](#), and [Xapian::NumberValueRangeProcessor](#).

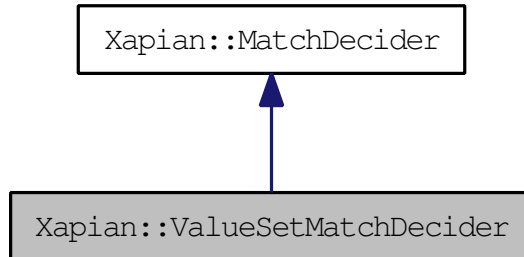
The documentation for this struct was generated from the following file:

- [xapian/queryparser.h](#)

## 7.68 Xapian::ValueSetMatchDecider Class Reference

[MatchDecider](#) filtering results based on whether document values are in a user-defined set.

Inheritance diagram for Xapian::ValueSetMatchDecider:



### Public Member Functions

- [ValueSetMatchDecider](#) ([Xapian::valueno](#) slot, bool inclusive\_)  
*Construct a [ValueSetMatchDecider](#).*
- void [add\\_value](#) (const std::string &value)  
*Add a value to the test set.*
- void [remove\\_value](#) (const std::string &value)  
*Remove a value from the test set.*
- bool [operator\(\)](#) (const [Xapian::Document](#) &doc) const  
*Decide whether we want this document to be in the [MSet](#).*

### 7.68.1 Detailed Description

[MatchDecider](#) filtering results based on whether document values are in a user-defined set.

### 7.68.2 Constructor & Destructor Documentation

#### 7.68.2.1 Xapian::ValueSetMatchDecider::ValueSetMatchDecider (Xapian::valueno slot, bool inclusive\_) [inline]

Construct a [ValueSetMatchDecider](#).

#### Parameters:

*slot* The value slot number to look in.

*inclusive\_* If true, match decider accepts documents which have a value in the specified slot which is a member of the test set; if false, match decider accepts documents which do not have a value in the specified slot.

### 7.68.3 Member Function Documentation

#### 7.68.3.1 void Xapian::ValueSetMatchDecider::add\_value (const std::string &value) [inline]

Add a value to the test set.

**Parameters:**

*value* The value to add to the test set.

#### 7.68.3.2 bool Xapian::ValueSetMatchDecider::operator() (const Xapian::Document &doc) const [virtual]

Decide whether we want this document to be in the [MSet](#).

Return true if the document is acceptable, or false if the document should be excluded from the [MSet](#).

Implements [Xapian::MatchDecider](#).

#### 7.68.3.3 void Xapian::ValueSetMatchDecider::remove\_value (const std::string &value) [inline]

Remove a value from the test set.

**Parameters:**

*value* The value to remove from the test set.

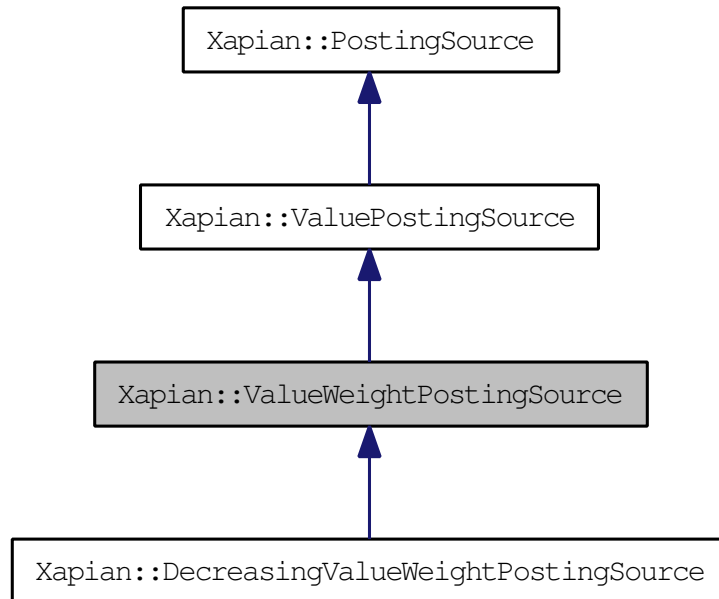
The documentation for this class was generated from the following file:

- [xapian/valuesetmatchdecider.h](#)

## 7.69 Xapian::ValueWeightPostingSource Class Reference

A posting source which reads weights from a value slot.

Inheritance diagram for Xapian::ValueWeightPostingSource:



### Public Member Functions

- [ValueWeightPostingSource](#) ([Xapian::valueno](#) slot\_)  
*Construct a [ValueWeightPostingSource](#).*
- [Xapian::weight](#) [get\\_weight](#) () const  
*Return the weight contribution for the current document.*
- [ValueWeightPostingSource](#) \* [clone](#) () const  
*Clone the posting source.*
- std::string [name](#) () const  
*Name of the posting source class.*
- std::string [serialise](#) () const  
*Serialise object parameters into a string.*
- [ValueWeightPostingSource](#) \* [unserialise](#) (const std::string &s) const  
*Create object given string serialisation returned by [serialise\(\)](#).*

- void [init](#) (const [Database](#) &db\_)  
*Set this [PostingSource](#) to the start of the list of postings.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*

### 7.69.1 Detailed Description

A posting source which reads weights from a value slot.

This returns entries for all documents in the given database which have a non empty values in the specified slot. It returns a weight calculated by applying `sortable_unserialise` to the value stored in the slot (so the values stored should probably have been calculated by applying `sortable_serialise` to a floating point number at index time).

The upper bound on the weight returned is set using the upper bound on the values in the specified slot, or `DBL_MAX` if value bounds aren't supported by the current backend.

For efficiency, this posting source doesn't check that the stored values are valid in any way, so it will never raise an exception due to invalid stored values. In particular, it doesn't ensure that the unserialised values are positive, which is a requirement for weights. The behaviour if the slot contains values which unserialise to negative values is undefined.

### 7.69.2 Constructor & Destructor Documentation

#### 7.69.2.1 Xapian::ValueWeightPostingSource::ValueWeightPostingSource (Xapian::valueno slot\_)

Construct a [ValueWeightPostingSource](#).

##### Parameters:

*slot\_* The value slot to read values from.

### 7.69.3 Member Function Documentation

#### 7.69.3.1 ValueWeightPostingSource\* Xapian::ValueWeightPostingSource::clone () const [virtual]

Clone the posting source.

The clone should inherit the configuration of the parent, but need not inherit the state. ie, the clone does not need to be in the same iteration position as the original: the

matcher will always call [init\(\)](#) on the clone before attempting to move the iterator, or read the information about the current position of the iterator.

This may return NULL to indicate that cloning is not supported. In this case, the [PostingSource](#) may only be used with a single-database search.

The default implementation returns NULL.

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

### 7.69.3.2 `std::string Xapian::ValueWeightPostingSource::get_description ()` `const` [virtual]

Return a string describing this object.

This default implementation returns a generic answer. This default is provided to avoid forcing those deriving their own [PostingSource](#) subclass from having to implement this (they may not care what [get\\_description\(\)](#) gives for their subclass).

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

### 7.69.3.3 `Xapian::weight Xapian::ValueWeightPostingSource::get_weight ()` `const` [virtual]

Return the weight contribution for the current document.

This default implementation always returns 0, for convenience when implementing "weight-less" [PostingSource](#) subclasses.

This method may assume that it will only be called when there is a "current document". In detail: [Xapian](#) will always call [init\(\)](#) on a [PostingSource](#) before calling this for the first time. It will also only call this if the [PostingSource](#) reports that it is pointing to a valid document (ie, it will not call it before calling at least one of [next\(\)](#), [skip\\_to\(\)](#) or [check\(\)](#), and will ensure that the [PostingSource](#) is not at the end by calling [at\\_end\(\)](#)).

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

### 7.69.3.4 `void Xapian::ValueWeightPostingSource::init (const Database & db)` [virtual]

Set this [PostingSource](#) to the start of the list of postings.

This is called automatically by the matcher prior to each query being processed.

If a [PostingSource](#) is used for multiple searches, [init\(\)](#) will therefore be called multiple times, and must handle this by using the database passed in the most recent call.

**Parameters:**

*db* The database which the [PostingSource](#) should iterate through.

Note: the database supplied to this method must not be modified: in particular, the `reopen()` method should not be called on it.

Note: in the case of a multi-database search, a separate [PostingSource](#) will be used for each database (the separate PostingSources will be obtained using `clone()`), and each [PostingSource](#) will be passed one of the sub-databases as the *db* parameter here. The *db* parameter will therefore always refer to a single database. All docids passed to, or returned from, the [PostingSource](#) refer to docids in that single database, rather than in the multi-database.

Reimplemented from [Xapian::ValuePostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

**7.69.3.5 `std::string Xapian::ValueWeightPostingSource::name () const`  
[virtual]**

Name of the posting source class.

This is used when serialising and unserialising posting sources; for example, for performing remote searches.

If the subclass is in a C++ namespace, the namespace should be included in the name, using "::" as a separator. For example, for a [PostingSource](#) subclass called "FooPostingSource" in the "Xapian" namespace the result of this call should be "Xapian::FooPostingSource".

This should only be implemented if [serialise\(\)](#) and [unserialise\(\)](#) are also implemented. The default implementation returns an empty string.

If this returns an empty string, [Xapian](#) will assume that [serialise\(\)](#) and [unserialise\(\)](#) are not implemented.

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

**7.69.3.6 `std::string Xapian::ValueWeightPostingSource::serialise () const`  
[virtual]**

Serialise object parameters into a string.

The serialised parameters should represent the configuration of the posting source, but need not (indeed, should not) represent the current iteration state.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

### 7.69.3.7 ValueWeightPostingSource\*

**Xapian::ValueWeightPostingSource::unserialise**  
(const std::string & s) const [virtual]

Create object given string serialisation returned by [serialise\(\)](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

#### Parameters:

- s A serialised instance of this [PostingSource](#) subclass.

Reimplemented from [Xapian::PostingSource](#).

Reimplemented in [Xapian::DecreasingValueWeightPostingSource](#).

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

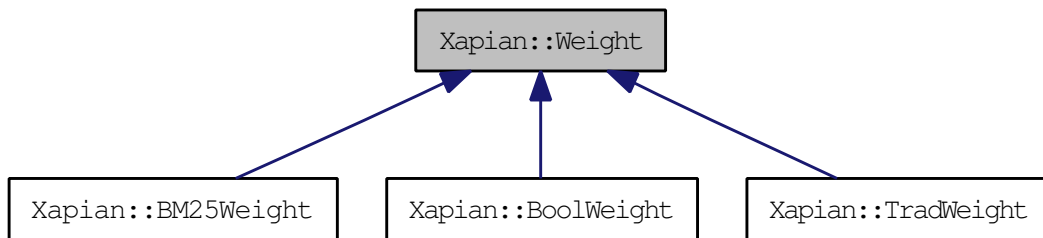
- [xapian/postingsource.h](#)



## 7.70 Xapian::Weight Class Reference

Abstract base class for weighting schemes.

Inheritance diagram for Xapian::Weight:



### Public Member Functions

- virtual `~Weight()`  
*Virtual destructor; because we have virtual methods.*
- virtual `Weight * clone() const = 0`  
*Clone this object.*
- virtual `std::string name() const`  
*Return the name of this weighting scheme.*
- virtual `std::string serialise() const`  
*Return this object's parameters serialised as a single string.*
- virtual `Weight * unserialise(const std::string &s) const`  
*Unserialise parameters.*
- virtual `Xapian::weight get_sumpart(Xapian::termcount wdf, Xapian::termcount doclen) const = 0`  
*Calculate the weight contribution for this object's term to a document.*
- virtual `Xapian::weight get_maxpart() const = 0`  
*Return an upper bound on what `get_sumpart()` can return for any document.*
- virtual `Xapian::weight get_sumextra(Xapian::termcount doclen) const = 0`  
*Calculate the term-independent weight component for a document.*
- virtual `Xapian::weight get_maxextra() const = 0`  
*Return an upper bound on what `get_sumextra()` can return for any document.*

## Protected Types

- enum [stat\\_flags](#)  
*Stats which the weighting scheme can use (see [need\\_stat\(\)](#)).*

## Protected Member Functions

- void [need\\_stat](#) ([stat\\_flags](#) flag)  
*Tell [Xapian](#) that your subclass will want a particular statistic.*
- virtual void [init](#) (double factor)=0  
*Allow the subclass to perform any initialisation it needs to.*
- [Weight](#) (const [Weight](#) &)  
*Don't allow copying.*
- [Weight](#) ()  
*Default constructor, needed by subclass constructors.*
- [Xapian::doccount](#) [get\\_collection\\_size](#) () const  
*The number of documents in the collection.*
- [Xapian::doccount](#) [get\\_rset\\_size](#) () const  
*The number of documents marked as relevant.*
- [Xapian::doclength](#) [get\\_average\\_length](#) () const  
*The average length of a document in the collection.*
- [Xapian::doccount](#) [get\\_termfreq](#) () const  
*The number of documents which this term indexes.*
- [Xapian::doccount](#) [get\\_retermfreq](#) () const  
*The number of relevant documents which this term indexes.*
- [Xapian::termcount](#) [get\\_query\\_length](#) () const  
*The length of the query.*
- [Xapian::termcount](#) [get\\_wqf](#) () const  
*The within-query-frequency of this term.*
- [Xapian::termcount](#) [get\\_doclength\\_upper\\_bound](#) () const  
*An lower bound on the maximum length of any document in the database.*
- [Xapian::termcount](#) [get\\_doclength\\_lower\\_bound](#) () const  
*An upper bound on the maximum length of any document in the database.*

- [Xapian::termcount get\\_wdf\\_upper\\_bound \(\)](#) const

*An upper bound on the wdf of this term.*

### 7.70.1 Detailed Description

Abstract base class for weighting schemes.

### 7.70.2 Constructor & Destructor Documentation

#### 7.70.2.1 virtual Xapian::Weight::~~Weight () [virtual]

Virtual destructor, because we have virtual methods.

#### 7.70.2.2 Xapian::Weight::Weight (const Weight &) [protected]

Don't allow copying.

This would ideally be private, but that causes a compilation error with GCC 4.1 (which appears to be a bug).

### 7.70.3 Member Function Documentation

#### 7.70.3.1 virtual Weight\* Xapian::Weight::clone () const [pure virtual]

Clone this object.

This method allocates and returns a copy of the object it is called on.

If your subclass is called FooWeight and has parameters a and b, then you would implement FooWeight::clone() like so:

```
FooWeight * FooWeight::clone() const { return new FooWeight(a, b); }
```

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

#### 7.70.3.2 Xapian::termcount Xapian::Weight::get\_doclength\_lower\_bound () const [inline, protected]

An upper bound on the maximum length of any document in the database.

This should only be used by [get\\_maxpart\(\)](#) and [get\\_maxextra\(\)](#).

### 7.70.3.3 **Xapian::termcount Xapian::Weight::get\_doclength\_upper\_bound ()** **const** [inline, protected]

An lower bound on the maximum length of any document in the database.

This should only be used by [get\\_maxpart\(\)](#) and [get\\_maxextra\(\)](#).

### 7.70.3.4 **virtual Xapian::weight Xapian::Weight::get\_maxextra () const** [pure virtual]

Return an upper bound on what [get\\_sumextra\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implemented in [Xapian::BoolWeight](#), [Xapian::BM25Weight](#), and [Xapian::TradWeight](#).

### 7.70.3.5 **virtual Xapian::weight Xapian::Weight::get\_maxpart () const** [pure virtual]

Return an upper bound on what [get\\_sumpart\(\)](#) can return for any document.

This information is used by the matcher to perform various optimisations, so strive to make the bound as tight as possible.

Implemented in [Xapian::BoolWeight](#), [Xapian::BM25Weight](#), and [Xapian::TradWeight](#).

### 7.70.3.6 **virtual Xapian::weight Xapian::Weight::get\_sumextra** **(Xapian::termcount doclen) const** [pure virtual]

Calculate the term-independent weight component for a document.

The parameter gives information about the document which may be used in the calculations:

#### Parameters:

*doclen* The document's length (unnormalised).

Implemented in [Xapian::BoolWeight](#), [Xapian::BM25Weight](#), and [Xapian::TradWeight](#).

### 7.70.3.7 **virtual Xapian::weight Xapian::Weight::get\_sumpart** **(Xapian::termcount wdf, Xapian::termcount doclen) const** [pure virtual]

Calculate the weight contribution for this object's term to a document.

The parameters give information about the document which may be used in the calculations:

**Parameters:**

*wdf* The within document frequency of the term in the document.

*doclen* The document's length (unnormalised).

Implemented in [Xapian::BoolWeight](#), [Xapian::BM25Weight](#), and [Xapian::TradWeight](#).

**7.70.3.8 Xapian::termcount Xapian::Weight::get\_wdf\_upper\_bound () const**  
[inline, protected]

An upper bound on the wdf of this term.

This should only be used by [get\\_maxpart\(\)](#) and [get\\_maxextra\(\)](#).

**7.70.3.9 virtual void Xapian::Weight::init (double factor)** [protected, pure virtual]

Allow the subclass to perform any initialisation it needs to.

**Parameters:**

*factor* Any scaling factor (e.g. from OP\_SCALE\_WEIGHT).

**7.70.3.10 virtual std::string Xapian::Weight::name () const** [virtual]

Return the name of this weighting scheme.

This name is used by the remote backend. It is passed along with the serialised parameters to the remote server so that it knows which class to create.

Return the full namespace-qualified name of your class here - if your class is called FooWeight, return "FooWeight" from this method ([Xapian::BM25Weight](#) returns "Xapian::BM25Weight" here).

If you don't want to support the remote backend, you can use the default implementation which simply returns an empty string.

Reimplemented in [Xapian::BoolWeight](#), [Xapian::BM25Weight](#), and [Xapian::TradWeight](#).

**7.70.3.11 void Xapian::Weight::need\_stat (stat\_flags flag)** [inline, protected]

Tell [Xapian](#) that your subclass will want a particular statistic.

Some of the statistics can be costly to fetch or calculate, so [Xapian](#) needs to know which are actually going to be used. You should call [need\\_stat\(\)](#) from your constructor for each such statistic.

**Parameters:**

*flag* The `stat_flags` value for a required statistic.

**7.70.3.12** `virtual std::string Xapian::Weight::serialise () const` [virtual]

Return this object's parameters serialised as a single string.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Reimplemented in [Xapian::BoolWeight](#), [Xapian::BM25Weight](#), and [Xapian::TradWeight](#).

**7.70.3.13** `virtual Weight* Xapian::Weight::unserialise (const std::string & s) const` [virtual]

Unserialise parameters.

This method unserialises parameters serialised by the [serialise\(\)](#) method and allocates and returns a new object initialised with them.

If you don't want to support the remote backend, you can use the default implementation which simply throws [Xapian::UnimplementedError](#).

Note that the returned object will be deallocated by [Xapian](#) after use with "delete". It must therefore have been allocated with "new".

Reimplemented in [Xapian::BoolWeight](#), [Xapian::BM25Weight](#), and [Xapian::TradWeight](#).

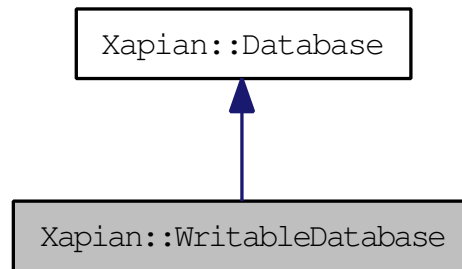
The documentation for this class was generated from the following file:

- [xapian/weight.h](#)

## 7.71 Xapian::WritableDatabase Class Reference

This class provides read/write access to a database.

Inheritance diagram for Xapian::WritableDatabase:



### Public Member Functions

- virtual [~WritableDatabase](#) ()  
*Destroy this handle on the database.*
- [WritableDatabase](#) ()  
*Create an empty [WritableDatabase](#).*
- [WritableDatabase](#) (const std::string &path, int action)  
*Open a database for update, automatically determining the database backend to use.*
- [WritableDatabase](#) (const [WritableDatabase](#) &other)  
*Copying is allowed.*
- void [operator=](#) (const [WritableDatabase](#) &other)  
*Assignment is allowed.*
- void [commit](#) ()  
*Commit any pending modifications made to the database.*
- void [flush](#) ()  
*Pre-1.1.0 name for [commit](#)().*
- void [begin\\_transaction](#) (bool flushed=true)  
*Begin a transaction.*
- void [commit\\_transaction](#) ()  
*Complete the transaction currently in progress.*
- void [cancel\\_transaction](#) ()

*Abort the transaction currently in progress, discarding the pending modifications made to the database.*

- [Xapian::docid add\\_document](#) (const [Xapian::Document](#) &document)  
*Add a new document to the database.*
- void [delete\\_document](#) ([Xapian::docid](#) did)  
*Delete a document from the database.*
- void [delete\\_document](#) (const std::string &unique\_term)  
*Delete any documents indexed by a term from the database.*
- void [replace\\_document](#) ([Xapian::docid](#) did, const [Xapian::Document](#) &document)  
*Replace a given document in the database.*
- [Xapian::docid replace\\_document](#) (const std::string &unique\_term, const [Xapian::Document](#) &document)  
*Replace any documents matching a term.*
- void [add\\_spelling](#) (const std::string &word, [Xapian::termcount](#) freqinc=1) const  
*Add a word to the spelling dictionary.*
- void [remove\\_spelling](#) (const std::string &word, [Xapian::termcount](#) freqdec=1) const  
*Remove a word from the spelling dictionary.*
- void [add\\_synonym](#) (const std::string &term, const std::string &synonym) const  
*Add a synonym for a term.*
- void [remove\\_synonym](#) (const std::string &term, const std::string &synonym) const  
*Remove a synonym for a term.*
- void [clear\\_synonyms](#) (const std::string &term) const  
*Remove all synonyms for a term.*
- void [set\\_metadata](#) (const std::string &key, const std::string &value)  
*Set the user-specified metadata associated with a given key.*
- std::string [get\\_description](#) () const  
*Return a string describing this object.*



### 7.71.1 Detailed Description

This class provides read/write access to a database.

### 7.71.2 Constructor & Destructor Documentation

#### 7.71.2.1 virtual Xapian::WritableDatabase::~~WritableDatabase () [virtual]

Destroy this handle on the database.

If there are no copies of this object remaining, the database will be closed. If there are any transactions in progress these will be aborted as if `cancel_transaction` had been called.

#### 7.71.2.2 Xapian::WritableDatabase::WritableDatabase (const std::string & *path*, int *action*)

Open a database for update, automatically determining the database backend to use.

If the database is to be created, [Xapian](#) will try to create the directory indicated by *path* if it doesn't already exist (but only the leaf directory, not recursively).

#### Parameters:

*path* directory that the database is stored in.

*action* one of:

- [Xapian::DB\\_CREATE\\_OR\\_OPEN](#) open for read/write; create if no db exists
- [Xapian::DB\\_CREATE](#) create new database; fail if db exists
- [Xapian::DB\\_CREATE\\_OR\\_OVERWRITE](#) overwrite existing db; create if none exists
- [Xapian::DB\\_OPEN](#) open for read/write; fail if no db exists

#### Exceptions:

[Xapian::DatabaseCorruptError](#) will be thrown if the database is in a corrupt state.

[Xapian::DatabaseLockError](#) will be thrown if a lock couldn't be acquired on the database.

#### 7.71.2.3 Xapian::WritableDatabase::WritableDatabase (const WritableDatabase & *other*)

Copying is allowed.

The internals are reference counted, so copying is cheap.

### 7.71.3 Member Function Documentation

#### 7.71.3.1 `Xapian::docid Xapian::WritableDatabase::add_document (const Xapian::Document & document)`

Add a new document to the database.

This method adds the specified document to the database, returning a newly allocated document ID. Automatically allocated document IDs come from a per-database monotonically increasing counter, so IDs from deleted documents won't be reused.

If you want to specify the document ID to be used, you should call [replace\\_document\(\)](#) instead.

Note that changes to the database won't be immediately committed to disk; see [commit\(\)](#) for more details.

As with all database modification operations, the effect is atomic: the document will either be fully added, or the document fails to be added and an exception is thrown (possibly at a later time when [commit\(\)](#) is called or the database is closed).

##### Parameters:

*document* The new document to be added.

##### Returns:

The document ID of the newly added document.

##### Exceptions:

[Xapian::DatabaseError](#) will be thrown if a problem occurs while writing to the database.

[Xapian::DatabaseCorruptError](#) will be thrown if the database is in a corrupt state.

#### 7.71.3.2 `void Xapian::WritableDatabase::add_spelling (const std::string & word, Xapian::termcount freqinc = 1) const`

Add a word to the spelling dictionary.

If the word is already present, its frequency is increased.

##### Parameters:

*word* The word to add.

*freqinc* How much to increase its frequency by (default 1).

#### 7.71.3.3 `void Xapian::WritableDatabase::add_synonym (const std::string & term, const std::string & synonym) const`

Add a synonym for a term.

If *synonym* is already a synonym for *term*, then no action is taken.

#### 7.71.3.4 void Xapian::WritableDatabase::begin\_transaction (bool *flushed* = true)

Begin a transaction.

In [Xapian](#) a transaction is a group of modifications to the database which are linked such that either all will be applied simultaneously or none will be applied at all. Even in the case of a power failure, this characteristic should be preserved (as long as the filesystem isn't corrupted, etc).

A transaction is started with [begin\\_transaction\(\)](#) and can either be committed by calling [commit\\_transaction\(\)](#) or aborted by calling [cancel\\_transaction\(\)](#).

By default, a transaction implicitly calls [commit\(\)](#) before and after so that the modifications stand and fall without affecting modifications before or after.

The downside of these implicit calls to [commit\(\)](#) is that small transactions can harm indexing performance in the same way that explicitly calling [commit\(\)](#) frequently can.

If you're applying atomic groups of changes and only wish to ensure that each group is either applied or not applied, then you can prevent the automatic [commit\(\)](#) before and after the transaction by starting the transaction with [begin\\_transaction\(false\)](#). However, if [cancel\\_transaction](#) is called (or if [commit\\_transaction](#) isn't called before the [WritableDatabase](#) object is destroyed) then any changes which were pending before the transaction began will also be discarded.

Transactions aren't currently supported by the [InMemory](#) backend.

#### Exceptions:

[Xapian::UnimplementedError](#) will be thrown if transactions are not available for this database type.

[Xapian::InvalidOperationError](#) will be thrown if this is called at an invalid time, such as when a transaction is already in progress.

#### 7.71.3.5 void Xapian::WritableDatabase::cancel\_transaction ()

Abort the transaction currently in progress, discarding the pending modifications made to the database.

If an error occurs in this method, an exception will be thrown, but the transaction will be cancelled anyway.

#### Exceptions:

[Xapian::DatabaseError](#) will be thrown if a problem occurs while modifying the database.

[Xapian::DatabaseCorruptError](#) will be thrown if the database is in a corrupt state.

*Xapian::InvalidOperationError* will be thrown if a transaction is not currently in progress.

*Xapian::UnimplementedError* will be thrown if transactions are not available for this database type.

#### 7.71.3.6 void Xapian::WritableDatabase::clear\_synonyms (const std::string & term) const

Remove all synonyms for a term.

If *term* has no synonyms, no action is taken.

#### 7.71.3.7 void Xapian::WritableDatabase::commit ()

Commit any pending modifications made to the database.

For efficiency reasons, when performing multiple updates to a database it is best (indeed, almost essential) to make as many modifications as memory will permit in a single pass through the database. To ensure this, *Xapian* batches up modifications.

This method may be called at any time to commit any pending modifications to the database.

If any of the modifications fail, an exception will be thrown and the database will be left in a state in which each separate addition, replacement or deletion operation has either been fully performed or not performed at all: it is then up to the application to work out which operations need to be repeated.

It's not valid to call *commit()* within a transaction.

Beware of calling *commit()* too frequently: this will make indexing take much longer.

Note that *commit()* need not be called explicitly: it will be called automatically when the database is closed, or when a sufficient number of modifications have been made. By default, this is every 10000 documents added, deleted, or modified. This value is rather conservative, and if you have a machine with plenty of memory, you can improve indexing throughput dramatically by setting *XAPIAN\_FLUSH\_THRESHOLD* in the environment to a larger value.

#### Exceptions:

*Xapian::DatabaseError* will be thrown if a problem occurs while modifying the database.

*Xapian::DatabaseCorruptError* will be thrown if the database is in a corrupt state.

#### 7.71.3.8 void Xapian::WritableDatabase::commit\_transaction ()

Complete the transaction currently in progress.

If this method completes successfully and this is a flushed transaction, all the database modifications made during the transaction will have been committed to the database.

If an error occurs, an exception will be thrown, and none of the modifications made to the database during the transaction will have been applied to the database.

In all cases the transaction will no longer be in progress.

**Exceptions:**

*Xapian::DatabaseError* will be thrown if a problem occurs while modifying the database.

*Xapian::DatabaseCorruptError* will be thrown if the database is in a corrupt state.

*Xapian::InvalidOperationError* will be thrown if a transaction is not currently in progress.

*Xapian::UnimplementedError* will be thrown if transactions are not available for this database type.

**7.71.3.9 void Xapian::WritableDatabase::delete\_document (const std::string &unique\_term)**

Delete any documents indexed by a term from the database.

This method removes any documents indexed by the specified term from the database.

A major use is for convenience when UIDs from another system are mapped to terms in *Xapian*, although this method has other uses (for example, you could add a "deletion date" term to documents at index time and use this method to delete all documents due for deletion on a particular date).

**Parameters:**

*unique\_term* The term to remove references to.

**Exceptions:**

*Xapian::DatabaseError* will be thrown if a problem occurs while writing to the database.

*Xapian::DatabaseCorruptError* will be thrown if the database is in a corrupt state.

**7.71.3.10 void Xapian::WritableDatabase::delete\_document (Xapian::docid did)**

Delete a document from the database.

This method removes the document with the specified document ID from the database.

Note that changes to the database won't be immediately committed to disk; see [commit\(\)](#) for more details.

As with all database modification operations, the effect is atomic: the document will either be fully removed, or the document fails to be removed and an exception is thrown (possibly at a later time when [commit\(\)](#) is called or the database is closed).

#### Parameters:

*did* The document ID of the document to be removed.

#### Exceptions:

[Xapian::DatabaseError](#) will be thrown if a problem occurs while writing to the database.

[Xapian::DatabaseCorruptError](#) will be thrown if the database is in a corrupt state.

#### 7.71.3.11 void Xapian::WritableDatabase::flush () [inline]

Pre-1.1.0 name for [commit\(\)](#).

Use [commit\(\)](#) instead in new code. This alias may be deprecated in the future.

#### 7.71.3.12 void Xapian::WritableDatabase::operator= (const WritableDatabase & other)

Assignment is allowed.

The internals are reference counted, so assignment is cheap.

Note that only an [WritableDatabase](#) may be assigned to an [WritableDatabase](#): an attempt to assign a [Database](#) is caught at compile-time.

#### 7.71.3.13 void Xapian::WritableDatabase::remove\_spelling (const std::string & word, Xapian::termcount freqdec = 1) const

Remove a word from the spelling dictionary.

The word's frequency is decreased, and if would become zero or less then the word is removed completely.

#### Parameters:

*word* The word to remove.

*freqdec* How much to decrease its frequency by (default 1).

**7.71.3.14 void Xapian::WritableDatabase::remove\_synonym (const std::string & term, const std::string & synonym) const**

Remove a synonym for a term.

If *synonym* isn't a synonym for *term*, then no action is taken.

**7.71.3.15 Xapian::docid Xapian::WritableDatabase::replace\_document (const std::string & unique\_term, const Xapian::Document & document)**

Replace any documents matching a term.

This method replaces any documents indexed by the specified term with the specified document. If any documents are indexed by the term, the lowest document ID will be used for the document, otherwise a new document ID will be generated as for `add_document`.

One common use is to allow UUIDs from another system to easily be mapped to terms in Xapian. Note that this method doesn't automatically add `unique_term` as a term, so you'll need to call `document.add_term(unique_term)` first when using `replace_document()` in this way.

Another possible use is to allow groups of documents to be marked for later deletion - for example, you could add a "deletion date" term to documents at index time and use this method to easily and efficiently delete all documents due for deletion on a particular date.

Note that changes to the database won't be immediately committed to disk; see `commit()` for more details.

As with all database modification operations, the effect is atomic: the document(s) will either be fully replaced, or the document(s) fail to be replaced and an exception is thrown (possibly at a later time when `commit()` is called or the database is closed).

**Parameters:**

*unique\_term* The "unique" term.

*document* The new document.

**Returns:**

The document ID that document was given.

**Exceptions:**

[\*Xapian::DatabaseError\*](#) will be thrown if a problem occurs while writing to the database.

[\*Xapian::DatabaseCorruptError\*](#) will be thrown if the database is in a corrupt state.

### 7.71.3.16 void Xapian::WritableDatabase::replace\_document (Xapian::docid *did*, const Xapian::Document & *document*)

Replace a given document in the database.

This method replaces the document with the specified document ID. If document ID *did* isn't currently used, the document will be added with document ID *did*.

The monotonic counter used for automatically allocating document IDs is increased so that the next automatically allocated document ID will be *did* + 1. Be aware that if you use this method to specify a high document ID for a new document, and also use [WritableDatabase::add\\_document\(\)](#), [Xapian](#) may get to a state where this counter wraps around and will be unable to automatically allocate document IDs!

Note that changes to the database won't be immediately committed to disk; see [commit\(\)](#) for more details.

As with all database modification operations, the effect is atomic: the document will either be fully replaced, or the document fails to be replaced and an exception is thrown (possibly at a later time when [commit\(\)](#) is called or the database is closed).

#### Parameters:

*did* The document ID of the document to be replaced.

*document* The new document.

#### Exceptions:

[Xapian::DatabaseError](#) will be thrown if a problem occurs while writing to the database.

[Xapian::DatabaseCorruptError](#) will be thrown if the database is in a corrupt state.

### 7.71.3.17 void Xapian::WritableDatabase::set\_metadata (const std::string & *key*, const std::string & *value*)

Set the user-specified metadata associated with a given key.

This method sets the metadata value associated with a given key. If there is already a metadata value stored in the database with the same key, the old value is replaced. If you want to delete an existing item of metadata, just set its value to the empty string.

User-specified metadata allows you to store arbitrary information in the form of (key,tag) pairs.

There's no hard limit on the number of metadata items, or the size of the metadata values. Metadata keys have a limited length, which depends on the backend. We recommend limiting them to 200 bytes. Empty keys are not valid, and specifying one will cause an exception.

Metadata modifications are committed to disk in the same way as modifications to the documents in the database are: i.e., modifications are atomic, and won't be committed to disk immediately (see [commit\(\)](#) for more details). This allows metadata to be used



to link databases with versioned external resources by storing the appropriate version number in a metadata item.

You can also use the metadata to store arbitrary extra information associated with terms, documents, or postings by encoding the termname and/or document id into the metadata key.

**Parameters:**

- key* The key of the metadata item to set.  
*value* The value of the metadata item to set.

**Exceptions:**

- Xapian::DatabaseError* will be thrown if a problem occurs while writing to the database.  
*Xapian::DatabaseCorruptError* will be thrown if the database is in a corrupt state.  
*Xapian::InvalidArgumentError* will be thrown if the key supplied is empty.  
*Xapian::UnimplementedError* will be thrown if the database backend in use doesn't support user-specified metadata.

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

- [xapian/database.h](#)



## Chapter 8

# File Documentation

### 8.1 xapian/error.h File Reference

Hierarchy of classes which [Xapian](#) can throw as exceptions.

#### Classes

- class [Xapian::Error](#)  
*All exceptions thrown by [Xapian](#) are subclasses of [Xapian::Error](#).*
- class [Xapian::LogicError](#)  
*The base class for exceptions indicating errors in the program logic.*
- class [Xapian::RuntimeError](#)  
*The base class for exceptions indicating errors only detectable at runtime.*
- class [Xapian::AssertionError](#)  
*[AssertionError](#) is thrown if a logical assertion inside [Xapian](#) fails.*
- class [Xapian::InvalidArgumentError](#)  
*[InvalidArgumentError](#) indicates an invalid parameter value was passed to the API.*
- class [Xapian::InvalidOperationError](#)  
*[InvalidOperationError](#) indicates the API was used in an invalid way.*
- class [Xapian::UnimplementedError](#)  
*[UnimplementedError](#) indicates an attempt to use an unimplemented feature.*
- class [Xapian::DatabaseError](#)  
*[DatabaseError](#) indicates some sort of database related error.*

- class [Xapian::DatabaseCorruptError](#)  
*DatabaseCorruptError* indicates database corruption was detected.
- class [Xapian::DatabaseCreateError](#)  
*DatabaseCreateError* indicates a failure to create a database.
- class [Xapian::DatabaseLockError](#)  
*DatabaseLockError* indicates failure to lock a database.
- class [Xapian::DatabaseModifiedError](#)  
*DatabaseModifiedError* indicates a database was modified.
- class [Xapian::DatabaseOpeningError](#)  
*DatabaseOpeningError* indicates failure to open a database.
- class [Xapian::DatabaseVersionError](#)  
*DatabaseVersionError* indicates that a database is in an unsupported format.
- class [Xapian::DocNotFoundError](#)  
*Indicates an attempt to access a document not present in the database.*
- class [Xapian::FeatureUnavailableError](#)  
*Indicates an attempt to use a feature which is unavailable.*
- class [Xapian::InternalError](#)  
*InternalError* indicates a runtime problem of some sort.
- class [Xapian::NetworkError](#)  
*Indicates a problem communicating with a remote database.*
- class [Xapian::NetworkTimeoutError](#)  
*Indicates a timeout expired while communicating with a remote database.*
- class [Xapian::QueryParserError](#)  
*Indicates a query string can't be parsed.*
- class [Xapian::SerialisationError](#)  
*Indicates an error in the std::string serialisation of an object.*
- class [Xapian::RangeError](#)  
*RangeError* indicates an attempt to access outside the bounds of a container.

## Namespaces

- namespace [Xapian](#)

*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### 8.1.1 Detailed Description

Hierarchy of classes which [Xapian](#) can throw as exceptions.

## 8.2 xapian/version.h File Reference

Define preprocessor symbols for the library version.

### Defines

- #define [XAPIAN\\_ENABLE\\_VISIBILITY](#)  
*The library was compiled with GCC's -fvisibility=hidden option.*
- #define [XAPIAN\\_VERSION](#) "1.2.6"  
*The version of [Xapian](#) as a C string literal.*
- #define [XAPIAN\\_MAJOR\\_VERSION](#) 1  
*The major component of the [Xapian](#) version.*
- #define [XAPIAN\\_MINOR\\_VERSION](#) 2  
*The minor component of the [Xapian](#) version.*
- #define [XAPIAN\\_REVISION](#) 6  
*The revision component of the [Xapian](#) version.*
- #define [XAPIAN\\_HAS\\_BRASS\\_BACKEND](#) 1  
*XAPIAN\_HAS\_BRASS\_BACKEND Defined if the brass backend is enabled.*
- #define [XAPIAN\\_HAS\\_CHERT\\_BACKEND](#) 1  
*XAPIAN\_HAS\_CHERT\_BACKEND Defined if the chert backend is enabled.*
- #define [XAPIAN\\_HAS\\_FLINT\\_BACKEND](#) 1  
*XAPIAN\_HAS\_FLINT\_BACKEND Defined if the flint backend is enabled.*
- #define [XAPIAN\\_HAS\\_INMEMORY\\_BACKEND](#) 1  
*XAPIAN\_HAS\_INMEMORY\_BACKEND Defined if the inmemory backend is enabled.*
- #define [XAPIAN\\_HAS\\_REMOTE\\_BACKEND](#) 1  
*XAPIAN\_HAS\_REMOTE\_BACKEND Defined if the remote backend is enabled.*

### 8.2.1 Detailed Description

Define preprocessor symbols for the library version.

## 8.2.2 Define Documentation

### 8.2.2.1 `#define XAPIAN_MAJOR_VERSION 1`

The major component of the [Xapian](#) version.

E.g. for [Xapian](#) 1.0.14 this would be: 1

### 8.2.2.2 `#define XAPIAN_MINOR_VERSION 2`

The minor component of the [Xapian](#) version.

E.g. for [Xapian](#) 1.0.14 this would be: 0

### 8.2.2.3 `#define XAPIAN_REVISION 6`

The revision component of the [Xapian](#) version.

E.g. for [Xapian](#) 1.0.14 this would be: 14

## 8.3 xapian.h File Reference

Public interfaces for the [Xapian](#) library.

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Functions

- `const char * Xapian::version\_string ()`  
*Report the version string of the library which the program is linked with.*
- `int Xapian::major\_version ()`  
*Report the major version of the library which the program is linked with.*
- `int Xapian::minor\_version ()`  
*Report the minor version of the library which the program is linked with.*
- `int Xapian::revision ()`  
*Report the revision of the library which the program is linked with.*

#### 8.3.1 Detailed Description

Public interfaces for the [Xapian](#) library.



## 8.4 xapian/compactor.h File Reference

Compact a database, or merge and compact several.

### Classes

- class [Xapian::Compactor](#)  
*Compact a database, or merge and compact several.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.4.1 Detailed Description

Compact a database, or merge and compact several.

## 8.5 xapian/database.h File Reference

API for working with [Xapian](#) databases.

### Classes

- class [Xapian::Database](#)  
*This class is used to access a database, or a group of databases.*
- class [Xapian::WritableDatabase](#)  
*This class provides read/write access to a database.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Variables

- const int [Xapian::DB\\_CREATE\\_OR\\_OPEN](#) = 1  
*Open for read/write; create if no db exists.*
- const int [Xapian::DB\\_CREATE](#) = 2  
*Create a new database; fail if db exists.*
- const int [Xapian::DB\\_CREATE\\_OR\\_OVERWRITE](#) = 3  
*Overwrite existing db; create if none exists.*
- const int [Xapian::DB\\_OPEN](#) = 4  
*Open for read/write; fail if no db exists.*

#### 8.5.1 Detailed Description

API for working with [Xapian](#) databases.

## 8.6 xapian/dbfactory.h File Reference

Factory functions for constructing Database and WritableDatabase objects.

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*
- namespace [Xapian::Auto](#)  
*[Database](#) factory functions which determine the database type automatically.*
- namespace [Xapian::InMemory](#)  
*[Database](#) factory functions for the inmemory backend.*
- namespace [Xapian::Brass](#)  
*[Database](#) factory functions for the brass backend.*
- namespace [Xapian::Chert](#)  
*[Database](#) factory functions for the chert backend.*
- namespace [Xapian::Flint](#)  
*[Database](#) factory functions for the flint backend.*
- namespace [Xapian::Remote](#)  
*[Database](#) factory functions for the remote backend.*

### Functions

- Database [Xapian::Auto::open\\_stub](#) (const std::string &file)  
*Construct a [Database](#) object for a stub database file.*
- WritableDatabase [Xapian::Auto::open\\_stub](#) (const std::string &file, int action)  
*Construct a [WritableDatabase](#) object for a stub database file.*
- WritableDatabase [Xapian::InMemory::open](#) ()  
*Construct a [WritableDatabase](#) object for a new, empty [InMemory](#) database.*
- Database [Xapian::Brass::open](#) (const std::string &dir)  
*Construct a [Database](#) object for read-only access to a [Brass](#) database.*
- WritableDatabase [Xapian::Brass::open](#) (const std::string &dir, int action, int block\_size=8192)  
*Construct a [Database](#) object for update access to a [Brass](#) database.*

- Database `Xapian::Chert::open` (const std::string &dir)  
Construct a *Database* object for read-only access to a *Chert* database.
- WritableDatabase `Xapian::Chert::open` (const std::string &dir, int action, int block\_size=8192)  
Construct a *Database* object for update access to a *Chert* database.
- Database `Xapian::Flint::open` (const std::string &dir)  
Construct a *Database* object for read-only access to a *Flint* database.
- WritableDatabase `Xapian::Flint::open` (const std::string &dir, int action, int block\_size=8192)  
Construct a *Database* object for update access to a *Flint* database.
- Database `Xapian::Remote::open` (const std::string &host, unsigned int port, `Xapian::timeout` timeout=10000, `Xapian::timeout` connect\_timeout=10000)  
Construct a *Database* object for read-only access to a remote database accessed via a TCP connection.
- WritableDatabase `Xapian::Remote::open_writable` (const std::string &host, unsigned int port, `Xapian::timeout` timeout=0, `Xapian::timeout` connect\_timeout=10000)  
Construct a *WritableDatabase* object for update access to a remote database accessed via a TCP connection.
- Database `Xapian::Remote::open` (const std::string &program, const std::string &args, `Xapian::timeout` timeout=10000)  
Construct a *Database* object for read-only access to a remote database accessed via a program.
- WritableDatabase `Xapian::Remote::open_writable` (const std::string &program, const std::string &args, `Xapian::timeout` timeout=0)  
Construct a *WritableDatabase* object for update access to a remote database accessed via a program.

### 8.6.1 Detailed Description

Factory functions for constructing Database and WritableDatabase objects.

## 8.7 xapian/document.h File Reference

API for working with documents.

### Classes

- class [Xapian::Document](#)  
*A handle representing a document in a [Xapian](#) database.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.7.1 Detailed Description

API for working with documents.

## 8.8 xapian/enquire.h File Reference

API for running queries.

### Classes

- class [Xapian::MSet](#)  
*A match set ([MSet](#)).*
- class [Xapian::MSetIterator](#)  
*An iterator pointing to items in an [MSet](#).*
- class [Xapian::ESet](#)  
*Class representing an ordered set of expand terms (an [ESet](#)).*
- class [Xapian::ESetIterator](#)  
*Iterate through terms in the [ESet](#).*
- class [Xapian::RSet](#)  
*A relevance set ([R-Set](#)).*
- class [Xapian::MatchDecider](#)  
*Base class for matcher decision functor.*
- class [Xapian::Enquire](#)  
*This class provides an interface to the information retrieval system for the purpose of searching.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Functions

- bool [Xapian::operator==](#) (const [MSetIterator](#) &a, const [MSetIterator](#) &b)  
*Equality test for [MSetIterator](#) objects.*
- bool [Xapian::operator!=](#) (const [MSetIterator](#) &a, const [MSetIterator](#) &b)  
*Inequality test for [MSetIterator](#) objects.*
- bool [Xapian::operator==](#) (const [ESetIterator](#) &a, const [ESetIterator](#) &b)  
*Equality test for [ESetIterator](#) objects.*

- bool [Xapian::operator!=](#) (const ESetIterator &a, const ESetIterator &b)  
*Inequality test for [ESetIterator](#) objects.*

### 8.8.1 Detailed Description

API for running queries.

## 8.9 xapian/errorhandler.h File Reference

Decide if a [Xapian::Error](#) exception should be ignored.

### Classes

- class [Xapian::ErrorHandler](#)  
*Decide if a [Xapian::Error](#) exception should be ignored.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.9.1 Detailed Description

Decide if a [Xapian::Error](#) exception should be ignored.



## 8.10 xapian/expanddecider.h File Reference

Allow rejection of terms during ESet generation.

### Classes

- class [Xapian::ExpandDecider](#)  
*Virtual base class for expand decider functor.*
- class [Xapian::ExpandDeciderAnd](#)  
*[ExpandDecider](#) subclass which rejects terms using two [ExpandDeciders](#).*
- class [Xapian::ExpandDeciderFilterTerms](#)  
*[ExpandDecider](#) subclass which rejects terms in a specified list.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.10.1 Detailed Description

Allow rejection of terms during ESet generation.

## 8.11 xapian/keymaker.h File Reference

Build key strings for MSet ordering or collapsing.

### Classes

- class [Xapian::KeyMaker](#)  
*Virtual base class for key making functors.*
- class [Xapian::MultiValueKeyMaker](#)  
*[KeyMaker](#) subclass which combines several values.*
- class [Xapian::Sorter](#)  
*Virtual base class for sorter functor.*
- class [Xapian::MultiValueSorter](#)  
*[Sorter](#) subclass which sorts by a several values.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.11.1 Detailed Description

Build key strings for MSet ordering or collapsing.

## 8.12 xapian/matchspy.h File Reference

MatchSpy implementation.

### Classes

- class [Xapian::MatchSpy](#)  
*Abstract base class for match spies.*
- class [Xapian::ValueCountMatchSpy](#)  
*Class for counting the frequencies of values in the matching documents.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.12.1 Detailed Description

MatchSpy implementation.

## 8.13 xapian/positioniterator.h File Reference

Classes for iterating through position lists.

### Classes

- class [Xapian::PositionIterator](#)  
*An iterator pointing to items in a list of positions.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Functions

- bool [Xapian::operator==](#) (const PositionIterator &a, const PositionIterator &b)  
*Test equality of two PositionIterators.*
- bool [Xapian::operator!=](#) (const PositionIterator &a, const PositionIterator &b)  
*Test inequality of two PositionIterators.*

#### 8.13.1 Detailed Description

Classes for iterating through position lists.

## 8.14 xapian/postingiterator.h File Reference

Classes for iterating through posting lists.

### Classes

- class [Xapian::PostingIterator](#)  
*An iterator pointing to items in a list of postings.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Functions

- bool [Xapian::operator==](#) (const PostingIterator &a, const PostingIterator &b)  
*Test equality of two PostingIterators.*
- bool [Xapian::operator!=](#) (const PostingIterator &a, const PostingIterator &b)  
*Test inequality of two PostingIterators.*

#### 8.14.1 Detailed Description

Classes for iterating through posting lists.

## 8.15 xapian/postingsource.h File Reference

External sources of posting information.

### Classes

- class [Xapian::PostingSource](#)  
*Base class which provides an "external" source of postings.*
- class [Xapian::ValuePostingSource](#)  
*A posting source which generates weights from a value slot.*
- class [Xapian::ValueWeightPostingSource](#)  
*A posting source which reads weights from a value slot.*
- class [Xapian::DecreasingValueWeightPostingSource](#)  
*Read weights from a value which is known to decrease as docid increases.*
- class [Xapian::ValueMapPostingSource](#)  
*A posting source which looks up weights in a map using values as the key.*
- class [Xapian::FixedWeightPostingSource](#)  
*A posting source which returns a fixed weight for all documents.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.15.1 Detailed Description

External sources of posting information.

## 8.16 xapian/query.h File Reference

Classes for representing a query.

### Classes

- class [Xapian::Query](#)  
*Class representing a query.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### 8.16.1 Detailed Description

Classes for representing a query.

## 8.17 xapian/queryparser.h File Reference

parsing a user query string to build a [Xapian::Query](#) object

### Classes

- class [Xapian::Stopper](#)  
*Base class for stop-word decision functor.*
- class [Xapian::SimpleStopper](#)  
*Simple implementation of [Stopper](#) class - this will suit most users.*
- struct [Xapian::ValueRangeProcessor](#)  
*Base class for value range processors.*
- class [Xapian::StringValueRangeProcessor](#)  
*Handle a string range.*
- class [Xapian::DateValueRangeProcessor](#)  
*Handle a date range.*
- class [Xapian::NumberValueRangeProcessor](#)  
*Handle a number range.*
- class [Xapian::QueryParser](#)  
*Build a [Xapian::Query](#) object from a user query string.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Functions

- `std::string Xapian::sortable\_serialise (double value)`  
*Convert a floating point number to a string, preserving sort order.*
- `double Xapian::sortable\_unserialise (const std::string &value)`  
*Convert a string encoded using [sortable\\_serialise](#) back to a floating point number.*



### **8.17.1 Detailed Description**

parsing a user query string to build a [Xapian::Query](#) object

## 8.18 xapian/registry.h File Reference

Class for looking up user subclasses during unserialisation.

### Classes

- class [Xapian::Registry](#)  
*Registry for user subclasses.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.18.1 Detailed Description

Class for looking up user subclasses during unserialisation.

## 8.19 xapian/stem.h File Reference

stemming algorithms

### Classes

- struct [Xapian::StemImplementation](#)  
*Class representing a stemming algorithm implementation.*
- class [Xapian::Stem](#)  
*Class representing a stemming algorithm.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.19.1 Detailed Description

stemming algorithms

## 8.20 xapian/termgenerator.h File Reference

parse free text and generate terms

### Classes

- class [Xapian::TermGenerator](#)  
*Parses a piece of text and generate terms.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.20.1 Detailed Description

parse free text and generate terms

## 8.21 xapian/termiterator.h File Reference

Classes for iterating through term lists.

### Classes

- class [Xapian::TermIterator](#)  
*An iterator pointing to items in a list of terms.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Functions

- bool [Xapian::operator==](#) (const TermIterator &a, const TermIterator &b)  
*Equality test for [TermIterator](#) objects.*
- bool [Xapian::operator!=](#) (const TermIterator &a, const TermIterator &b)  
*Inequality test for [TermIterator](#) objects.*

#### 8.21.1 Detailed Description

Classes for iterating through term lists.

## 8.22 xapian/types.h File Reference

typedefs for [Xapian](#)

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Typedefs

- typedef unsigned [Xapian::doccount](#)  
*A count of documents.*
- typedef int [Xapian::doccount\\_diff](#)  
*A signed difference between two counts of documents.*
- typedef unsigned [Xapian::docid](#)  
*A unique identifier for a document.*
- typedef double [Xapian::doclength](#)  
*A normalised document length.*
- typedef int [Xapian::percent](#)  
*The percentage score for a document in an [MSet](#).*
- typedef unsigned [Xapian::termcount](#)  
*A counts of terms.*
- typedef int [Xapian::termcount\\_diff](#)  
*A signed difference between two counts of terms.*
- typedef unsigned [Xapian::termpos](#)  
*A term position within a document or query.*
- typedef int [Xapian::termpos\\_diff](#)  
*A signed difference between two term positions.*
- typedef unsigned [Xapian::timeout](#)  
*A timeout value in milliseconds.*
- typedef unsigned [Xapian::valueno](#)  
*The number for a value slot in a document.*

- typedef int [Xapian::valueno\\_diff](#)  
*A signed difference between two value slot numbers.*
- typedef double [Xapian::weight](#)  
*The weight of a document or term.*

## Variables

- const valueno [Xapian::BAD\\_VALUENO](#) = static\_cast<valueno>(-1)  
*Reserved value to indicate "no valueno".*

### 8.22.1 Detailed Description

typedefs for [Xapian](#)

## 8.23 xapian/unicode.h File Reference

Unicode and UTF-8 related classes and functions.

### Classes

- class [Xapian::Utf8Iterator](#)  
*An iterator which returns [Unicode](#) character values from a UTF-8 encoded string.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*
- namespace [Xapian::Unicode](#)  
*Functions associated with handling [Unicode](#) characters.*

### Enumerations

- enum [Xapian::Unicode::category](#)  
*Each Unicode character is in exactly one of these categories.*

### Functions

- unsigned [Xapian::Unicode::nonascii\\_to\\_utf8](#) (unsigned ch, char \*buf)  
*Convert a single non-ASCII [Unicode](#) character to UTF-8.*
- unsigned [Xapian::Unicode::to\\_utf8](#) (unsigned ch, char \*buf)  
*Convert a single [Unicode](#) character to UTF-8.*
- void [Xapian::Unicode::append\\_utf8](#) (std::string &s, unsigned ch)  
*Append the UTF-8 representation of a single [Unicode](#) character to a std::string.*
- category [Xapian::Unicode::get\\_category](#) (unsigned ch)  
*Return the category which a given [Unicode](#) character falls into.*
- bool [Xapian::Unicode::is\\_wordchar](#) (unsigned ch)  
*Test if a given [Unicode](#) character is "word character".*
- bool [Xapian::Unicode::is\\_whitespace](#) (unsigned ch)  
*Test if a given [Unicode](#) character is a whitespace character.*



- bool [Xapian::Unicode::is\\_currency](#) (unsigned ch)  
*Test if a given [Unicode](#) character is a currency symbol.*
- unsigned [Xapian::Unicode::tolower](#) (unsigned ch)  
*Convert a [Unicode](#) character to lowercase.*
- unsigned [Xapian::Unicode::toupper](#) (unsigned ch)  
*Convert a [Unicode](#) character to uppercase.*
- std::string [Xapian::Unicode::tolower](#) (const std::string &term)  
*Convert a UTF-8 std::string to lowercase.*
- std::string [Xapian::Unicode::toupper](#) (const std::string &term)  
*Convert a UTF-8 std::string to uppercase.*

### 8.23.1 Detailed Description

Unicode and UTF-8 related classes and functions.

## 8.24 xapian/valueiterator.h File Reference

Class for iterating over document values.

### Classes

- class [Xapian::ValueIterator](#)  
*Class for iterating over document values.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

### Functions

- bool [Xapian::operator==](#) (const ValueIterator &a, const ValueIterator &b)  
*Equality test for [ValueIterator](#) objects.*
- bool [Xapian::operator!=](#) (const ValueIterator &a, const ValueIterator &b)  
*Inequality test for [ValueIterator](#) objects.*

#### 8.24.1 Detailed Description

Class for iterating over document values.

## 8.25 xapian/valuesetmatchdecider.h File Reference

MatchDecider subclass for filtering results by value.

### Classes

- class [Xapian::ValueSetMatchDecider](#)  
*MatchDecider filtering results based on whether document values are in a user-defined set.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.25.1 Detailed Description

MatchDecider subclass for filtering results by value.

## 8.26 xapian/weight.h File Reference

Weighting scheme API.

### Classes

- class [Xapian::Weight](#)  
*Abstract base class for weighting schemes.*
- class [Xapian::BoolWeight](#)  
*Class implementing a "boolean" weighting scheme.*
- class [Xapian::BM25Weight](#)  
*[Xapian::Weight](#) subclass implementing the BM25 probabilistic formula.*
- class [Xapian::TradWeight](#)  
*[Xapian::Weight](#) subclass implementing the traditional probabilistic formula.*

### Namespaces

- namespace [Xapian](#)  
*The [Xapian](#) namespace contains public interfaces for the [Xapian](#) library.*

#### 8.26.1 Detailed Description

Weighting scheme API.

# Index

- ~Database
  - Xapian::Database, [56](#)
- ~ExpandDecider
  - Xapian::ExpandDecider, [113](#)
- ~KeyMaker
  - Xapian::KeyMaker, [132](#)
- ~MatchSpy
  - Xapian::MatchSpy, [137](#)
- ~Query
  - Xapian::Query, [177](#)
- ~Weight
  - Xapian::Weight, [257](#)
- ~WritableDatabase
  - Xapian::WritableDatabase, [263](#)
- add\_boolean\_prefix
  - Xapian::QueryParser, [184](#)
- add\_boolean\_term
  - Xapian::Document, [88](#)
- add\_database
  - Xapian::Database, [56](#)
- add\_document
  - Xapian::WritableDatabase, [264](#)
- add\_mapping
  - Xapian::ValueMapPostingSource, [237](#)
- add\_matchspy
  - Xapian::Enquire, [95](#)
- add\_posting
  - Xapian::Document, [88](#)
- add\_prefix
  - Xapian::QueryParser, [184](#)
- add\_source
  - Xapian::Compactor, [49](#)
- add\_spelling
  - Xapian::WritableDatabase, [264](#)
- add\_synonym
  - Xapian::WritableDatabase, [264](#)
- add\_term
  - Xapian::Document, [89](#)
- add\_value
  - Xapian::Document, [89](#)
  - Xapian::ValueSetMatchDecider, [249](#)
- allterms\_begin
  - Xapian::Database, [56](#)
- AssertionError
  - Xapian::AssertionError, [40](#)
- assign
  - Xapian::Utf8Iterator, [225](#)
- at\_end
  - Xapian::FixedWeightPostingSource, [120](#)
  - Xapian::PostingSource, [167](#)
  - Xapian::ValuePostingSource, [243](#)
- BAD\_VALUENO
  - Xapian, [24](#)
- begin\_transaction
  - Xapian::WritableDatabase, [265](#)
- BM25Weight
  - Xapian::BM25Weight, [42](#)
- BoolWeight
  - Xapian::BoolWeight, [46](#)
- cancel\_transaction
  - Xapian::WritableDatabase, [265](#)
- category
  - Xapian::Unicode, [37](#)
- check
  - Xapian::DecreasingValueWeightPostingSource, [80](#)
  - Xapian::FixedWeightPostingSource, [121](#)
  - Xapian::PostingSource, [168](#)
  - Xapian::ValueIterator, [234](#)
  - Xapian::ValuePostingSource, [243](#)
- clear\_mappings
  - Xapian::ValueMapPostingSource, [237](#)
- clear\_synonyms
  - Xapian::WritableDatabase, [266](#)
- clone

- Xapian::DecreasingValueWeightPostingSource, 80
- Xapian::FixedWeightPostingSource, 121
- Xapian::MatchSpy, 137
- Xapian::PostingSource, 168
- Xapian::ValueCountMatchSpy, 229
- Xapian::ValueMapPostingSource, 237
- Xapian::ValueWeightPostingSource, 251
- Xapian::Weight, 257
- close
  - Xapian::Database, 56
- commit
  - Xapian::WritableDatabase, 266
- commit\_transaction
  - Xapian::WritableDatabase, 266
- convert\_to\_percent
  - Xapian::MSet, 142
- Database
  - Xapian::Database, 56
- DatabaseCorruptError
  - Xapian::DatabaseCorruptError, 63
- DatabaseCreateError
  - Xapian::DatabaseCreateError, 65
- DatabaseError
  - Xapian::DatabaseError, 66
- DatabaseLockError
  - Xapian::DatabaseLockError, 68
- DatabaseModifiedError
  - Xapian::DatabaseModifiedError, 70
- DatabaseOpeningError
  - Xapian::DatabaseOpeningError, 72
- DatabaseVersionError
  - Xapian::DatabaseVersionError, 74
- DateValueRangeProcessor
  - Xapian::DateValueRangeProcessor, 76
- DB\_CREATE
  - Xapian, 24
- DB\_CREATE\_OR\_OPEN
  - Xapian, 24
- DB\_CREATE\_OR\_OVERWRITE
  - Xapian, 24
- DB\_OPEN
  - Xapian, 24
- delete\_document
  - Xapian::WritableDatabase, 267
- Document
  - Xapian::Document, 88
- empty
  - Xapian::Query, 179
- Enquire
  - Xapian::Enquire, 94
- ExpandDeciderAnd
  - Xapian::ExpandDeciderAnd, 114
- ExpandDeciderFilterTerms
  - Xapian::ExpandDeciderFilterTerms, 115
- feature\_flag
  - Xapian::QueryParser, 182
- FeatureUnavailableError
  - Xapian::FeatureUnavailableError, 117
- fetch
  - Xapian::MSet, 142
- FixedWeightPostingSource
  - Xapian::FixedWeightPostingSource, 120
- FLAG\_AUTO\_MULTIWORD\_-SYNONYMS
  - Xapian::QueryParser, 183
- FLAG\_AUTO\_SYNONYMS
  - Xapian::QueryParser, 183
- FLAG\_BOOLEAN
  - Xapian::QueryParser, 183
- FLAG\_BOOLEAN\_ANY\_CASE
  - Xapian::QueryParser, 183
- FLAG\_DEFAULT
  - Xapian::QueryParser, 183
- FLAG\_LOVEHATE
  - Xapian::QueryParser, 183
- FLAG\_PARTIAL
  - Xapian::QueryParser, 183
- FLAG\_PHRASE
  - Xapian::QueryParser, 183
- doccount
  - Xapian, 21
- doccount\_diff
  - Xapian, 21
- docid
  - Xapian, 21
- doclength
  - Xapian, 21
- DocNotFoundError
  - Xapian::DocNotFoundError, 84

- FLAG\_PURE\_NOT
  - Xapian::QueryParser, 183
- FLAG\_SPELLING
  - Xapian::TermGenerator, 211
- FLAG\_SPELLING\_CORRECTION
  - Xapian::QueryParser, 183
- FLAG\_SYNONYM
  - Xapian::QueryParser, 183
- FLAG\_WILDCARD
  - Xapian::QueryParser, 183
- flags
  - Xapian::TermGenerator, 211
- flush
  - Xapian::WritableDatabase, 268
- get\_available\_languages
  - Xapian::Stem, 205
- get\_collapse\_count
  - Xapian::MSetIterator, 147
- get\_collection\_freq
  - Xapian::Database, 57
- get\_context
  - Xapian::Error, 107
- get\_corrected\_query\_string
  - Xapian::QueryParser, 185
- get\_data
  - Xapian::Document, 89
- get\_default\_op
  - Xapian::QueryParser, 185
- get\_description
  - Xapian::DecreasingValueWeightPostingSource, 80
  - Xapian::FixedWeightPostingSource, 122
  - Xapian::MatchSpy, 137
  - Xapian::PostingSource, 169
  - Xapian::ValueCountMatchSpy, 229
  - Xapian::ValueMapPostingSource, 238
  - Xapian::ValueWeightPostingSource, 252
- get\_docid
  - Xapian::Document, 89
  - Xapian::FixedWeightPostingSource, 122
  - Xapian::PostingSource, 169
  - Xapian::ValueIterator, 234
  - Xapian::ValuePostingSource, 244
- get\_doclength
  - Xapian::PostingIterator, 164
- get\_doclength\_lower\_bound
  - Xapian::Database, 57
  - Xapian::Weight, 257
- get\_doclength\_upper\_bound
  - Xapian::Weight, 257
- get\_document
  - Xapian::Database, 57
  - Xapian::MSetIterator, 148
- get\_ebound
  - Xapian::ESet, 110
- get\_error\_string
  - Xapian::Error, 107
- get\_eset
  - Xapian::Enquire, 95, 96
- get\_firstitem
  - Xapian::MSet, 143
- get\_length
  - Xapian::Query, 179
- get\_match\_spy
  - Xapian::Registry, 194
- get\_matches\_estimated
  - Xapian::MSet, 143
- get\_matches\_lower\_bound
  - Xapian::MSet, 143
- get\_matches\_upper\_bound
  - Xapian::MSet, 143
- get\_matching\_terms\_begin
  - Xapian::Enquire, 97
- get\_max\_attained
  - Xapian::MSet, 144
- get\_max\_possible
  - Xapian::MSet, 144
- get\_maxextra
  - Xapian::BM25Weight, 42
  - Xapian::BoolWeight, 46
  - Xapian::TradWeight, 218
  - Xapian::Weight, 258
- get\_maxpart
  - Xapian::BM25Weight, 42
  - Xapian::BoolWeight, 46
  - Xapian::TradWeight, 218
  - Xapian::Weight, 258
- get\_metadata
  - Xapian::Database, 58
- get\_mset
  - Xapian::Enquire, 98–100
- get\_percent
  - Xapian::MSetIterator, 148
- get\_posting\_source
  - Xapian::Registry, 194

- get\_query
  - Xapian::Enquire, [101](#)
- get\_rank
  - Xapian::MSetIterator, [148](#)
- get\_spelling\_suggestion
  - Xapian::Database, [58](#)
- get\_sumextra
  - Xapian::BM25Weight, [43](#)
  - Xapian::BoolWeight, [46](#)
  - Xapian::TradWeight, [218](#)
  - Xapian::Weight, [258](#)
- get\_sumpart
  - Xapian::BM25Weight, [43](#)
  - Xapian::BoolWeight, [46](#)
  - Xapian::TradWeight, [219](#)
  - Xapian::Weight, [258](#)
- get\_termfreq
  - Xapian::MSet, [144](#)
  - Xapian::TermIterator, [215](#)
- get\_termfreq\_est
  - Xapian::FixedWeightPostingSource, [122](#)
  - Xapian::PostingSource, [169](#)
  - Xapian::ValuePostingSource, [244](#)
- get\_termfreq\_max
  - Xapian::FixedWeightPostingSource, [122](#)
  - Xapian::PostingSource, [169](#)
  - Xapian::ValuePostingSource, [244](#)
- get\_termfreq\_min
  - Xapian::FixedWeightPostingSource, [122](#)
  - Xapian::PostingSource, [170](#)
  - Xapian::ValuePostingSource, [244](#)
- get\_terms\_begin
  - Xapian::Query, [179](#)
- get\_termweight
  - Xapian::MSet, [144](#)
- get\_total
  - Xapian::ValueCountMatchSpy, [230](#)
- get\_uuid
  - Xapian::Database, [58](#)
- get\_value
  - Xapian::Document, [89](#)
- get\_value\_freq
  - Xapian::Database, [59](#)
- get\_value\_lower\_bound
  - Xapian::Database, [59](#)
- get\_value\_upper\_bound
  - Xapian::Database, [59](#)
- get\_valueno
  - Xapian::ValueIterator, [234](#)
- get\_wdf
  - Xapian::TermIterator, [215](#)
- get\_wdf\_upper\_bound
  - Xapian::Weight, [259](#)
- get\_weight
  - Xapian::DecreasingValueWeightPostingSource, [81](#)
  - Xapian::FixedWeightPostingSource, [123](#)
  - Xapian::PostingSource, [170](#)
  - Xapian::ValueMapPostingSource, [238](#)
  - Xapian::ValueWeightPostingSource, [252](#)
- get\_weighting\_scheme
  - Xapian::Registry, [194](#)
- increase\_termpos
  - Xapian::TermGenerator, [212](#)
- index\_text
  - Xapian::TermGenerator, [212](#)
- index\_text\_without\_positions
  - Xapian::TermGenerator, [212](#)
- init
  - Xapian::DecreasingValueWeightPostingSource, [81](#)
  - Xapian::FixedWeightPostingSource, [123](#)
  - Xapian::PostingSource, [170](#)
  - Xapian::ValueMapPostingSource, [238](#)
  - Xapian::ValuePostingSource, [244](#)
  - Xapian::ValueWeightPostingSource, [252](#)
  - Xapian::Weight, [259](#)
- InternalError
  - Xapian::InternalError, [126](#)
- InvalidArgumentError
  - Xapian::InvalidArgumentError, [128](#)
- InvalidOperationError
  - Xapian::InvalidOperationError, [130](#)
- keep\_alive
  - Xapian::Database, [60](#)
- left
  - Xapian::Utf8Iterator, [226](#)
- major\_version



- Xapian, 23
- MatchAll
  - Xapian::Query, 180
- MatchNothing
  - Xapian::Query, 180
- max\_size
  - Xapian::ESet, 110
  - Xapian::MSet, 144
- merge\_results
  - Xapian::MatchSpy, 137
  - Xapian::ValueCountMatchSpy, 230
- metadata\_keys\_begin
  - Xapian::Database, 60
- minor\_version
  - Xapian, 23
- name
  - Xapian::BM25Weight, 43
  - Xapian::BoolWeight, 47
  - Xapian::DecreasingValueWeightPostingSource, 81
  - Xapian::FixedWeightPostingSource, 123
  - Xapian::MatchSpy, 138
  - Xapian::PostingSource, 171
  - Xapian::TradWeight, 219
  - Xapian::ValueCountMatchSpy, 230
  - Xapian::ValueMapPostingSource, 239
  - Xapian::ValueWeightPostingSource, 253
  - Xapian::Weight, 259
- need\_stat
  - Xapian::Weight, 259
- NetworkError
  - Xapian::NetworkError, 155
- NetworkTimeoutError
  - Xapian::NetworkTimeoutError, 157
- next
  - Xapian::DecreasingValueWeightPostingSource, 82
  - Xapian::FixedWeightPostingSource, 124
  - Xapian::PostingSource, 171
  - Xapian::ValuePostingSource, 245
- nonascii\_to\_utf8
  - Xapian::Unicode, 37
- NumberValueRangeProcessor
  - Xapian::NumberValueRangeProcessor, 159
- op
  - Xapian::Query, 176
- OP\_AND
  - Xapian::Query, 176
- OP\_AND\_MAYBE
  - Xapian::Query, 176
- OP\_AND\_NOT
  - Xapian::Query, 176
- OP\_ELITE\_SET
  - Xapian::Query, 176
- OP\_FILTER
  - Xapian::Query, 176
- OP\_NEAR
  - Xapian::Query, 176
- OP\_OR
  - Xapian::Query, 176
- OP\_PHRASE
  - Xapian::Query, 176
- OP\_SCALE\_WEIGHT
  - Xapian::Query, 176
- OP\_SYNONYM
  - Xapian::Query, 177
- OP\_VALUE\_GE
  - Xapian::Query, 176
- OP\_VALUE\_LE
  - Xapian::Query, 177
- OP\_VALUE\_RANGE
  - Xapian::Query, 176
- OP\_XOR
  - Xapian::Query, 176
- open
  - Xapian::Brass, 26
  - Xapian::Chert, 28
  - Xapian::Flint, 30
  - Xapian::InMemory, 32
  - Xapian::Remote, 33, 34
- open\_stub
  - Xapian::Auto, 25
- open\_writable
  - Xapian::Remote, 34
- operator\*
  - Xapian::Utf8Iterator, 226
- operator()
  - Xapian::DateValueRangeProcessor, 77
  - Xapian::ErrorHandler, 108
  - Xapian::KeyMaker, 132
  - Xapian::MatchDecider, 135
  - Xapian::MatchSpy, 138
  - Xapian::MultiValueKeyMaker, 150

- Xapian::MultiValueSorter, 153
- Xapian::NumberValueRangeProcessor, remove\_term  
159
- Xapian::Stem, 205
- Xapian::StringValueRangeProcessor, 209
- Xapian::ValueCountMatchSpy, 230
- Xapian::ValueRangeProcessor, 247
- Xapian::ValueSetMatchDecider, 249
- operator++
  - Xapian::Utf8Iterator, 226
- operator=
  - Xapian::Database, 60
  - Xapian::Document, 90
  - Xapian::PositionIterator, 162
  - Xapian::PostingIterator, 164
  - Xapian::Query, 179
  - Xapian::Registry, 194
  - Xapian::TermIterator, 215
  - Xapian::WritableDatabase, 268
- operator==
  - Xapian::Utf8Iterator, 226
- parse\_query
  - Xapian::QueryParser, 185
- percent
  - Xapian, 21
- PositionIterator
  - Xapian::PositionIterator, 161
- PostingIterator
  - Xapian::PostingIterator, 164
- postlist\_begin
  - Xapian::Database, 60
- Query
  - Xapian::Query, 177, 178
- QueryParserError
  - Xapian::QueryParserError, 189
- RangeError
  - Xapian::RangeError, 191
- raw
  - Xapian::Utf8Iterator, 226
- Registry
  - Xapian::Registry, 194
- remove\_posting
  - Xapian::Document, 90
- remove\_spelling
  - Xapian::WritableDatabase, 268
- remove\_synonym
  - Xapian::WritableDatabase, 268
  - Xapian::Document, 90
- remove\_value
  - Xapian::ValueSetMatchDecider, 249
- reopen
  - Xapian::Database, 60
- replace\_document
  - Xapian::WritableDatabase, 269
- resolve\_duplicate\_metadata
  - Xapian::Compactor, 49
- revision
  - Xapian, 23
- SerialisationError
  - Xapian::SerialisationError, 199
- serialise
  - Xapian::BM25Weight, 43
  - Xapian::BoolWeight, 47
  - Xapian::DecreasingValueWeightPostingSource, 82
  - Xapian::Document, 90
  - Xapian::FixedWeightPostingSource, 124
  - Xapian::MatchSpy, 138
  - Xapian::PostingSource, 171
  - Xapian::Query, 179
  - Xapian::TradWeight, 219
  - Xapian::ValueCountMatchSpy, 230
  - Xapian::ValueMapPostingSource, 239
  - Xapian::ValueWeightPostingSource, 253
  - Xapian::Weight, 260
- serialise\_results
  - Xapian::MatchSpy, 138
  - Xapian::ValueCountMatchSpy, 231
- set\_block\_size
  - Xapian::Compactor, 50
- set\_collapse\_key
  - Xapian::Enquire, 101
- set\_compaction\_level
  - Xapian::Compactor, 50
- set\_cutoff
  - Xapian::Enquire, 102
- set\_database
  - Xapian::QueryParser, 186
- set\_default\_op
  - Xapian::QueryParser, 186
- set\_default\_weight

- Xapian::ValueMapPostingSource, 239
- set\_destdir
  - Xapian::Compactor, 50
- set\_docid\_order
  - Xapian::Enquire, 102
- set\_flags
  - Xapian::TermGenerator, 213
- set\_max\_wildcard\_expansion
  - Xapian::QueryParser, 187
- set\_maxweight
  - Xapian::PostingSource, 172
- set\_metadata
  - Xapian::WritableDatabase, 270
- set\_multipass
  - Xapian::Compactor, 50
- set\_query
  - Xapian::Enquire, 102
- set\_renumber
  - Xapian::Compactor, 50
- set\_sort\_by\_key
  - Xapian::Enquire, 103
- set\_sort\_by\_key\_then\_relevance
  - Xapian::Enquire, 103
- set\_sort\_by\_relevance
  - Xapian::Enquire, 103
- set\_sort\_by\_relevance\_then\_key
  - Xapian::Enquire, 103
- set\_sort\_by\_relevance\_then\_value
  - Xapian::Enquire, 104
- set\_sort\_by\_value
  - Xapian::Enquire, 104
- set\_sort\_by\_value\_then\_relevance
  - Xapian::Enquire, 104
- set\_status
  - Xapian::Compactor, 51
- set\_stemmer
  - Xapian::QueryParser, 187
- set\_stemming\_strategy
  - Xapian::QueryParser, 187
- set\_stopper
  - Xapian::TermGenerator, 213
- set\_weighting\_scheme
  - Xapian::Enquire, 105
- skip\_to
  - Xapian::DecreasingValueWeightPostingSource, 82
  - Xapian::FixedWeightPostingSource, 124
  - Xapian::PositionIterator, 162
  - Xapian::PostingIterator, 165
  - Xapian::PostingSource, 172
  - Xapian::TermIterator, 215
  - Xapian::ValueIterator, 234
  - Xapian::ValuePostingSource, 245
- sortable\_serialise
  - Xapian, 23
- sortable\_unserialise
  - Xapian, 23
- spellings\_begin
  - Xapian::Database, 61
- Stem
  - Xapian::Stem, 204, 205
- StringValueRangeProcessor
  - Xapian::StringValueRangeProcessor, 208, 209
- synonym\_keys\_begin
  - Xapian::Database, 61
- synonyms\_begin
  - Xapian::Database, 61
- term\_exists
  - Xapian::Database, 61
- termcount
  - Xapian, 22
- termcount\_diff
  - Xapian, 22
- termfreq\_est
  - Xapian::ValuePostingSource, 246
- termfreq\_max
  - Xapian::ValuePostingSource, 246
- termfreq\_min
  - Xapian::ValuePostingSource, 246
- TermIterator
  - Xapian::TermIterator, 215
- termlist\_count
  - Xapian::Document, 91
- termpos\_diff
  - Xapian, 22
- timeout
  - Xapian, 22
- to\_utf8
  - Xapian::Unicode, 37
- top\_values\_begin
  - Xapian::ValueCountMatchSpy, 231
- trad\_weight
  - Xapian::TradWeight, 218
- UnimplementedError
  - Xapian::UnimplementedError, 221

- unserialise
  - Xapian::BM25Weight, [44](#)
  - Xapian::BoolWeight, [47](#)
  - Xapian::DecreasingValueWeightPostingSource, [83](#)
  - Xapian::FixedWeightPostingSource, [125](#)
  - Xapian::MatchSpy, [139](#)
  - Xapian::PostingSource, [173](#)
  - Xapian::Query, [179](#), [180](#)
  - Xapian::TradWeight, [219](#)
  - Xapian::ValueCountMatchSpy, [231](#)
  - Xapian::ValueMapPostingSource, [240](#)
  - Xapian::ValueWeightPostingSource, [253](#)
  - Xapian::Weight, [260](#)
- Utf8Iterator
  - Xapian::Utf8Iterator, [224](#), [225](#)
- ValueIterator
  - Xapian::ValueIterator, [234](#)
- ValueMapPostingSource
  - Xapian::ValueMapPostingSource, [237](#)
- valueno
  - Xapian, [22](#)
- valueno\_diff
  - Xapian, [22](#)
- ValuePostingSource
  - Xapian::ValuePostingSource, [243](#)
- values\_begin
  - Xapian::ValueCountMatchSpy, [231](#)
- ValueSetMatchDecider
  - Xapian::ValueSetMatchDecider, [248](#)
- ValueWeightPostingSource
  - Xapian::ValueWeightPostingSource, [251](#)
- version.h
  - XAPIAN\_MAJOR\_VERSION, [277](#)
  - XAPIAN\_MINOR\_VERSION, [277](#)
  - XAPIAN\_REVISION, [277](#)
- version\_string
  - Xapian, [24](#)
- Weight
  - Xapian::Weight, [257](#)
- weight
  - Xapian, [22](#)
- WritableDatabase
  - Xapian::WritableDatabase, [263](#)
- Xapian, [13](#)
- BAD\_VALUENO, [24](#)
- DB\_CREATE, [24](#)
- DB\_CREATE\_OR\_OPEN, [24](#)
- DB\_CREATE\_OR\_OVERWRITE, [24](#)
- DB\_OPEN, [24](#)
- doccount, [21](#)
- doccount\_diff, [21](#)
- docid, [21](#)
- doclength, [21](#)
- major\_version, [23](#)
- minor\_version, [23](#)
- percent, [21](#)
- revision, [23](#)
- sortable\_serialise, [23](#)
- sortable\_unserialise, [23](#)
- termcount, [22](#)
- termcount\_diff, [22](#)
- termpos\_diff, [22](#)
- timeout, [22](#)
- valueno, [22](#)
- valueno\_diff, [22](#)
- version\_string, [24](#)
- weight, [22](#)
- xapian.h, [278](#)
- xapian/compactor.h, [279](#)
- xapian/database.h, [280](#)
- xapian/dbfactory.h, [281](#)
- xapian/document.h, [283](#)
- xapian/enquire.h, [284](#)
- xapian/error.h, [273](#)
- xapian/errorhandler.h, [286](#)
- xapian/expanddecider.h, [287](#)
- xapian/keymaker.h, [288](#)
- xapian/matchspy.h, [289](#)
- xapian/positioniterator.h, [290](#)
- xapian/postingiterator.h, [291](#)
- xapian/postingsource.h, [292](#)
- xapian/query.h, [293](#)
- xapian/queryparser.h, [294](#)
- xapian/registry.h, [296](#)
- xapian/stem.h, [297](#)
- xapian/termgenerator.h, [298](#)
- xapian/termiterator.h, [299](#)
- xapian/types.h, [300](#)
- xapian/unicode.h, [302](#)
- xapian/valueiterator.h, [304](#)

- xapian/valuesetmatchdecider.h, 305
- xapian/version.h, 276
- xapian/weight.h, 306
- Xapian::AssertionError, 39
  - AssertionError, 40
- Xapian::Auto, 25
  - open\_stub, 25
- Xapian::BM25Weight, 41
  - BM25Weight, 42
  - get\_maxextra, 42
  - get\_maxpart, 42
  - get\_sumextra, 43
  - get\_sumpart, 43
  - name, 43
  - serialise, 43
  - unserialise, 44
- Xapian::BoolWeight, 45
  - BoolWeight, 46
  - get\_maxextra, 46
  - get\_maxpart, 46
  - get\_sumextra, 46
  - get\_sumpart, 46
  - name, 47
  - serialise, 47
  - unserialise, 47
- Xapian::Brass, 26
  - open, 26
- Xapian::Chert, 28
  - open, 28
- Xapian::Compactor, 49
  - add\_source, 49
  - resolve\_duplicate\_metadata, 49
  - set\_block\_size, 50
  - set\_compaction\_level, 50
  - set\_destdir, 50
  - set\_multipass, 50
  - set\_renumber, 50
  - set\_status, 51
- Xapian::Database, 52
  - ~Database, 56
  - add\_database, 56
  - allterms\_begin, 56
  - close, 56
  - Database, 56
  - get\_collection\_freq, 57
  - get\_doclength\_lower\_bound, 57
  - get\_document, 57
  - get\_metadata, 58
  - get\_spelling\_suggestion, 58
  - get\_uuid, 58
  - get\_value\_freq, 59
  - get\_value\_lower\_bound, 59
  - get\_value\_upper\_bound, 59
  - keep\_alive, 60
  - metadata\_keys\_begin, 60
  - operator=, 60
  - postlist\_begin, 60
  - reopen, 60
  - spellings\_begin, 61
  - synonym\_keys\_begin, 61
  - synonyms\_begin, 61
  - term\_exists, 61
- Xapian::DatabaseCorruptError, 62
  - DatabaseCorruptError, 63
- Xapian::DatabaseCreateError, 64
  - DatabaseCreateError, 65
- Xapian::DatabaseError, 66
  - DatabaseError, 66
- Xapian::DatabaseLockError, 67
  - DatabaseLockError, 68
- Xapian::DatabaseModifiedError, 69
  - DatabaseModifiedError, 70
- Xapian::DatabaseOpeningError, 71
  - DatabaseOpeningError, 72
- Xapian::DatabaseVersionError, 73
  - DatabaseVersionError, 74
- Xapian::DateValueRangeProcessor, 75
  - DateValueRangeProcessor, 76
  - operator(), 77
- Xapian::DecreasingValueWeightPostingSource, 78
  - check, 80
  - clone, 80
  - get\_description, 80
  - get\_weight, 81
  - init, 81
  - name, 81
  - next, 82
  - serialise, 82
  - skip\_to, 82
  - unserialise, 83
- Xapian::DocNotFoundError, 84
  - DocNotFoundError, 84
- Xapian::Document, 86
  - add\_boolean\_term, 88
  - add\_posting, 88
  - add\_term, 89
  - add\_value, 89
  - Document, 88
  - get\_data, 89

- get\_docid, 89
- get\_value, 89
- operator=, 90
- remove\_posting, 90
- remove\_term, 90
- serialise, 90
- termlist\_count, 91
- Xapian::Enquire, 92
  - add\_matchspy, 95
  - Enquire, 94
  - get\_eset, 95, 96
  - get\_matching\_terms\_begin, 97
  - get\_mset, 98–100
  - get\_query, 101
  - set\_collapse\_key, 101
  - set\_cutoff, 102
  - set\_docid\_order, 102
  - set\_query, 102
  - set\_sort\_by\_key, 103
  - set\_sort\_by\_key\_then\_relevance, 103
  - set\_sort\_by\_relevance, 103
  - set\_sort\_by\_relevance\_then\_key, 103
  - set\_sort\_by\_relevance\_then\_value, 104
  - set\_sort\_by\_value, 104
  - set\_sort\_by\_value\_then\_relevance, 104
  - set\_weighting\_scheme, 105
- Xapian::Error, 106
  - get\_context, 107
  - get\_error\_string, 107
- Xapian::ErrorHandler, 108
  - operator(), 108
- Xapian::ESet, 109
  - get\_ebound, 110
  - max\_size, 110
- Xapian::ESetIterator, 111
- Xapian::ExpandDecider, 113
  - ~ExpandDecider, 113
- Xapian::ExpandDeciderAnd, 114
  - ExpandDeciderAnd, 114
- Xapian::ExpandDeciderFilterTerms, 115
  - ExpandDeciderFilterTerms, 115
- Xapian::FeatureUnavailableError, 117
  - FeatureUnavailableError, 117
- Xapian::FixedWeightPostingSource, 119
  - at\_end, 120
  - check, 121
  - clone, 121
  - FixedWeightPostingSource, 120
  - get\_description, 122
  - get\_docid, 122
  - get\_termfreq\_est, 122
  - get\_termfreq\_max, 122
  - get\_termfreq\_min, 122
  - get\_weight, 123
  - init, 123
  - name, 123
  - next, 124
  - serialise, 124
  - skip\_to, 124
  - unserialise, 125
- Xapian::Flint, 30
  - open, 30
- Xapian::InMemory, 32
  - open, 32
- Xapian::InternalError, 126
  - InternalError, 126
- Xapian::InvalidArgumentError, 128
  - InvalidArgumentError, 128
- Xapian::InvalidOperationError, 130
  - InvalidOperationError, 130
- Xapian::KeyMaker, 132
  - ~KeyMaker, 132
  - operator(), 132
- Xapian::LogicError, 134
- Xapian::MatchDecider, 135
  - operator(), 135
- Xapian::MatchSpy, 136
  - ~MatchSpy, 137
  - clone, 137
  - get\_description, 137
  - merge\_results, 137
  - name, 138
  - operator(), 138
  - serialise, 138
  - serialise\_results, 138
  - unserialise, 139
- Xapian::MSet, 140
  - convert\_to\_percent, 142
  - fetch, 142
  - get\_firstitem, 143
  - get\_matches\_estimated, 143
  - get\_matches\_lower\_bound, 143
  - get\_matches\_upper\_bound, 143
  - get\_max\_attained, 144
  - get\_max\_possible, 144
  - get\_termfreq, 144

- get\_termweight, 144
- max\_size, 144
- Xapian::MSetIterator, 146
  - get\_collapse\_count, 147
  - get\_document, 148
  - get\_percent, 148
  - get\_rank, 148
- Xapian::MultiValueKeyMaker, 150
  - operator(), 150
- Xapian::MultiValueSorter, 152
  - operator(), 153
- Xapian::NetworkError, 154
  - NetworkError, 155
- Xapian::NetworkTimeoutError, 156
  - NetworkTimeoutError, 157
- Xapian::NumberValueRangeProcessor, 158
  - NumberValueRangeProcessor, 159
  - operator(), 159
- Xapian::PositionIterator, 161
  - operator=, 162
  - PositionIterator, 161
  - skip\_to, 162
- Xapian::PostingIterator, 163
  - get\_doclength, 164
  - operator=, 164
  - PostingIterator, 164
  - skip\_to, 165
- Xapian::PostingSource, 166
  - at\_end, 167
  - check, 168
  - clone, 168
  - get\_description, 169
  - get\_docid, 169
  - get\_termfreq\_est, 169
  - get\_termfreq\_max, 169
  - get\_termfreq\_min, 170
  - get\_weight, 170
  - init, 170
  - name, 171
  - next, 171
  - serialise, 171
  - set\_maxweight, 172
  - skip\_to, 172
  - unserialise, 173
- Xapian::Query, 174
  - ~Query, 177
  - empty, 179
  - get\_length, 179
  - get\_terms\_begin, 179
  - MatchAll, 180
  - MatchNothing, 180
  - op, 176
  - OP\_AND, 176
  - OP\_AND\_MAYBE, 176
  - OP\_AND\_NOT, 176
  - OP\_ELITE\_SET, 176
  - OP\_FILTER, 176
  - OP\_NEAR, 176
  - OP\_OR, 176
  - OP\_PHRASE, 176
  - OP\_SCALE\_WEIGHT, 176
  - OP\_SYNONYM, 177
  - OP\_VALUE\_GE, 176
  - OP\_VALUE\_LE, 177
  - OP\_VALUE\_RANGE, 176
  - OP\_XOR, 176
  - operator=, 179
  - Query, 177, 178
  - serialise, 179
  - unserialise, 179, 180
- Xapian::QueryParser, 181
  - add\_boolean\_prefix, 184
  - add\_prefix, 184
  - feature\_flag, 182
  - FLAG\_AUTO\_MULTIWORD\_-SYNONYMS, 183
  - FLAG\_AUTO\_SYNONYMS, 183
  - FLAG\_BOOLEAN, 183
  - FLAG\_BOOLEAN\_ANY\_CASE, 183
  - FLAG\_DEFAULT, 183
  - FLAG\_LOVEHATE, 183
  - FLAG\_PARTIAL, 183
  - FLAG\_PHRASE, 183
  - FLAG\_PURE\_NOT, 183
  - FLAG\_SPELLING\_-CORRECTION, 183
  - FLAG\_SYNONYM, 183
  - FLAG\_WILDCARD, 183
  - get\_corrected\_query\_string, 185
  - get\_default\_op, 185
  - parse\_query, 185
  - set\_database, 186
  - set\_default\_op, 186
  - set\_max\_wildcard\_expansion, 187
  - set\_stemmer, 187
  - set\_stemming\_strategy, 187
- Xapian::QueryParserError, 189
  - QueryParserError, 189

- Xapian::RangeError, 191
  - RangeError, 191
- Xapian::Registry, 193
  - get\_match\_spy, 194
  - get\_posting\_source, 194
  - get\_weighting\_scheme, 194
  - operator=, 194
  - Registry, 194
- Xapian::Remote, 33
  - open, 33, 34
  - open\_writable, 34
- Xapian::RSet, 196
- Xapian::RuntimeError, 198
- Xapian::SerialisationError, 199
  - SerialisationError, 199
- Xapian::SimpleStopper, 201
- Xapian::Sorter, 202
- Xapian::Stem, 203
  - get\_available\_languages, 205
  - operator(), 205
  - Stem, 204, 205
- Xapian::StemImplementation, 206
- Xapian::Stopper, 207
- Xapian::StringValueRangeProcessor, 208
  - operator(), 209
  - StringValueRangeProcessor, 208, 209
- Xapian::TermGenerator, 210
  - FLAG\_SPELLING, 211
  - flags, 211
  - increase\_termpos, 212
  - index\_text, 212
  - index\_text\_without\_positions, 212
  - set\_flags, 213
  - set\_stopper, 213
- Xapian::TermIterator, 214
  - get\_termfreq, 215
  - get\_wdf, 215
  - operator=, 215
  - skip\_to, 215
  - TermIterator, 215
- Xapian::TradWeight, 217
  - get\_maxextra, 218
  - get\_maxpart, 218
  - get\_sumextra, 218
  - get\_sumpart, 219
  - name, 219
  - serialise, 219
  - TradWeight, 218
  - unserialise, 219
- Xapian::Unicode, 36
  - category, 37
  - nonascii\_to\_utf8, 37
  - to\_utf8, 37
- Xapian::UnimplementedError, 221
  - UnimplementedError, 221
- Xapian::Utf8Iterator, 223
  - assign, 225
  - left, 226
  - operator\*, 226
  - operator++, 226
  - operator==, 226
  - raw, 226
  - Utf8Iterator, 224, 225
- Xapian::ValueCountMatchSpy, 228
  - clone, 229
  - get\_description, 229
  - get\_total, 230
  - merge\_results, 230
  - name, 230
  - operator(), 230
  - serialise, 230
  - serialise\_results, 231
  - top\_values\_begin, 231
  - unserialise, 231
  - values\_begin, 231
- Xapian::ValueIterator, 233
  - check, 234
  - get\_docid, 234
  - get\_valueno, 234
  - skip\_to, 234
  - ValueIterator, 234
- Xapian::ValueMapPostingSource, 236
  - add\_mapping, 237
  - clear\_mappings, 237
  - clone, 237
  - get\_description, 238
  - get\_weight, 238
  - init, 238
  - name, 239
  - serialise, 239
  - set\_default\_weight, 239
  - unserialise, 240
  - ValueMapPostingSource, 237
- Xapian::ValuePostingSource, 241
  - at\_end, 243
  - check, 243
  - get\_docid, 244
  - get\_termfreq\_est, 244
  - get\_termfreq\_max, 244



- get\_termfreq\_min, [244](#)
- init, [244](#)
- next, [245](#)
- skip\_to, [245](#)
- termfreq\_est, [246](#)
- termfreq\_max, [246](#)
- termfreq\_min, [246](#)
- ValuePostingSource, [243](#)
- Xapian::ValueRangeProcessor, [247](#)
  - operator(), [247](#)
- Xapian::ValueSetMatchDecider, [248](#)
  - add\_value, [249](#)
  - operator(), [249](#)
  - remove\_value, [249](#)
  - ValueSetMatchDecider, [248](#)
- Xapian::ValueWeightPostingSource, [250](#)
  - clone, [251](#)
  - get\_description, [252](#)
  - get\_weight, [252](#)
  - init, [252](#)
  - name, [253](#)
  - serialise, [253](#)
  - unserialise, [253](#)
  - ValueWeightPostingSource, [251](#)
- Xapian::Weight, [255](#)
  - ~Weight, [257](#)
  - clone, [257](#)
  - get\_doclength\_lower\_bound, [257](#)
  - get\_doclength\_upper\_bound, [257](#)
  - get\_maxextra, [258](#)
  - get\_maxpart, [258](#)
  - get\_sumextra, [258](#)
  - get\_sumpart, [258](#)
  - get\_wdf\_upper\_bound, [259](#)
  - init, [259](#)
  - name, [259](#)
  - need\_stat, [259](#)
  - serialise, [260](#)
  - unserialise, [260](#)
  - Weight, [257](#)
- Xapian::WritableDatabase, [261](#)
  - ~WritableDatabase, [263](#)
  - add\_document, [264](#)
  - add\_spelling, [264](#)
  - add\_synonym, [264](#)
  - begin\_transaction, [265](#)
  - cancel\_transaction, [265](#)
  - clear\_synonyms, [266](#)
  - commit, [266](#)
  - commit\_transaction, [266](#)
  - delete\_document, [267](#)
  - flush, [268](#)
  - operator=, [268](#)
  - remove\_spelling, [268](#)
  - remove\_synonym, [268](#)
  - replace\_document, [269](#)
  - set\_metadata, [270](#)
  - WritableDatabase, [263](#)
- XAPIAN\_MAJOR\_VERSION
  - version.h, [277](#)
- XAPIAN\_MINOR\_VERSION
  - version.h, [277](#)
- XAPIAN\_REVISION
  - version.h, [277](#)