
FCS-QL

Release 0.1

Erik Körner

Jul 12, 2023

CONTENTS

1	FCS-QL for Python	1
1.1	Installation	1
1.2	Building	1
1.3	Development	2
1.4	Build documentation	2
1.5	See also	3
2	Reference	5
2.1	fcsql	5
2.2	fcsql.parser	5
2.3	ANTLR generated parser	16
3	Indices and tables	19
	Python Module Index	21
	Index	23

FCS-QL FOR PYTHON

- CLARIN-FCS Core 2.0 query language grammar and parser
- based on [Github: clarin-eric/fcs-ql](#) and [Github: clarin-eric/fcs-simple-endpoint](#)
- for more details visit: [CLARIN FCS Technical Details](#)

1.1 Installation

```
# built package
python3 -m pip install dist/fcs_ql_parser-<version>-py2.py3-none-any.whl
# or
python3 -m pip install dist/fcs-ql-parser-<version>.tar.gz

# for local development
python3 -m pip install -e .
```

1.2 Building

Fetch (or update) grammar files:

```
git clone https://github.com/clarin-eric/fcs-ql.git
cp fcs-ql/src/main/antlr4/eu/clarin/sru/fcs/qlparser/*.g4 src/fcsql/
```

(Re-)Generate python parser code:

```
# create virtual env
python3 -m venv venv
source venv/bin/activate
pip install -U pip setuptools wheel

# install antler tool
python3 -m pip install antlr4-tools
# pip install -e .[antlr]

cd src/fcsql
antlr4 -Dlanguage=Python3 *.g4
```

Build package:

```
# pip install -e .[build]
python3 -m build
```

1.3 Development

- Uses pytest (with coverage, clarity and randomly plugins).

```
python3 -m pip install -e .[test]

pytest
```

Run style checks:

```
# general style checks
python3 -m pip install -e .[style]

black --check .
flake8 . --show-source --statistics
isort --check --diff .
mypy src

# building the package and check metadata
python3 -m pip install -e .[build]

python3 -m build
twine check --strict dist/*

# build documentation and check links ...
python3 -m pip install -e .[docs]

sphinx-build -b html docs dist/docs
sphinx-build -b linkcheck docs dist/docs
```

1.4 Build documentation

```
python3 -m pip install -r ./docs/requirements.txt
# or
python3 -m pip install -e .[docs]

sphinx-build -b html docs dist/docs
sphinx-build -b linkcheck docs dist/docs
```

1.5 See also

- [clarin-eric/fcq-ql](#)
- [clarin-eric/fcs-simple-endpoint](#)
- [Specification on CLARIN FCS 2.0](#)

REFERENCE

2.1 fcsql

exception fcsql.SyntaxError

class fcsql.ExceptionThrowingErrorListener

syntaxError(recognizer, offendingSymbol, line, column, msg, e)

fcsql.**antlr_parse**(input: *str*) → QueryContext

fcsql.**parse**(input: *str*) → *QueryNode*

Simple wrapper to generate a *QueryParser* and to parse some input string into a *QueryNode*.

Parameters

input – raw input query string

Returns

QueryNode – parsed query

Throws:

QueryParserException: if an error occurred

2.2 fcsql.parser

fcsql.parser.OCCURS_UNBOUNDED = -1

Atom occurrence if not bound.

class fcsql.parser.QueryNodeType(*value*)

Node types of FCS-QL expression tree nodes.

QUERY_SEGMENT = 'QuerySegment'

Segment query.

QUERY_GROUP = 'QueryGroup'

Group query.

QUERY_SEQUENCE = 'QuerySequence'

Sequence query.

QUERY_DISJUNCTION = 'QueryDisjunction'

Or query.

QUERY_WITH_WITHIN = 'QueryWithin'

Query with within part.

EXPRESSION = 'Expression'

Simple expression.

EXPRESSION_WILDCARD = 'Wildcard'

Wildcard expression.

EXPRESSION_GROUP = 'Group'

Group expression.

EXPRESSION_OR = 'Or'

Or expression.

EXPRESSION_AND = 'And'

And expression.

EXPRESSION_NOT = 'Not'

Not expression.

SIMPLE_WITHIN = 'SimpleWithin'

Simple within part.

class fcsql.parser.**Operator**(value)

FCS-QL operators.

EQUALS = 'Eq'

EQUALS operator.

NOT_EQUALS = 'Ne'

NOT-EQUALS operator.

class fcsql.parser.**RegexFlag**(value)

FCS-QL expression tree regex flags.

char: **str**

CASE_INSENSITIVE = 'case-insensitive'

Case insensitive.

CASE_SENSITIVE = 'case-sensitive'

Case sensitive.

LITERAL_MATCHING = 'literal-matching'

match exactly (= literally)

IGNORE_DIACRITICS = 'ignore-diacritics'

Ignore all diacritics.

class fcsql.parser.**SimpleWithinScope**(value)

The within scope.

SENTENCE = 'Sentence'

sentence scope (small)

UTTERANCE = 'Utterance'

utterance scope (small)

PARAGRAPH = 'Paragraph'
 paragraph scope (medium)

TURN = 'Turn'
 turn scope (medium)

TEXT = 'Text'
 text scope (large)

SESSION = 'Session'
 session scope (large)

class fcsql.parser.**QueryVisitor**

Interface implementing a Visitor pattern for FCS-QL expression trees.

Default method implementations do nothing.

visit(node: [QueryNode](#)) → None

Visit a query node. Generic handler, dispatches to visit methods based on *QueryNodeType* if exists else do nothing:

method = "visit_" + node.node_type.value

Parameters

node – the node to visit

Returns

None

class fcsql.parser.**QueryNode**(node_type: [QueryNodeType](#), children: [List\[QueryNode\]](#) | None = None, child: [QueryNode](#) | None = None)

Base class for FCS-QL expression tree nodes.

[Constructor]

Parameters

- **node_type** – the type of the node
- **children** – the children of this node or None. Defaults to None.
- **child** – the child of this node or None. Defaults to None.

node_type

The node type of this node.

parent: [QueryNode](#) | None

The parent node of this node.

None if this is the root node.

children

The children of this node.

has_node_type(node_type: [QueryNodeType](#)) → bool

Check, if node if of given type.

Parameters

node_type – type to check against

Returns

bool – True if node is of given type, False otherwise

Raises

TypeError – if node_type is None

property child_count: **int**

Get the number of children of this node.

Returns

int – the number of children of this node

get_child(*idx: int, clazz: Type[_T] | None = None*) → *QueryNode* | None

Get a child node of specified type by index.

When supplied with *clazz* parameter, only child nodes of the requested type are counted.

Parameters

- **idx** – the index of the child node (if *clazz* provided, only considers child nodes of requested type)
- **clazz** – the type to nodes to be considered, optional

Returns

QueryNode – the child node of this node or None if not child was found (e.g. type mismatch or index out of bounds)

get_first_child(*clazz: Type[_T] | None = None*) → *QueryNode* | None

Get this first child node.

Parameters

clazz – the type to nodes to be considered

Returns

QueryNode – the first child node of this node or None

get_last_child(*clazz: Type[_T] | None = None*) → *QueryNode* | None

Get this last child node.

Parameters

clazz – the type to nodes to be considered

Returns

QueryNode – the last child node of this node or None

abstract accept(*visitor: QueryVisitor*) → None

class fcsql.parser.**Expression**(*qualifier: str | None, identifier: str, operator: Operator, regex: str, regex_flags: Set[RegexFlag] | None*)

A FCS-QL expression tree SIMPLE expression node.

[Constructor]

Parameters

- **qualifier** – the layer identifier qualifier or None
- **identifier** – the layer identifier
- **operator** – the operator
- **regex** – the regular expression

- **regex_flags** – the regular expression flags or None

qualifier

The Layer Type Identifier qualifier.

None if not used in this expression.

identifier

The layer identifier.

operator

The operator.

regex

The regex value.

regex_flags

The regex flags set.

None if no flags were used in this expression.

has_layer_identifier(*identifier: str*) → bool

Check if the expression used a given **Layer Type Identifier**.

Parameters

identifier – the Layer Type Identifier to check against

Returns

bool – True if this identifier was used, False otherwise

Raises

TypeError – if identifier is None

is_layer_qualifier_empty() → bool

Check if the Layer Type Identifier qualifier is empty.

Returns

bool – True if no Layer Type Identifier qualifier was set, False otherwise

has_layer_qualifier(*qualifier: str*) → bool

Check if the expression used a given qualifier for the Layer Type Identifier.

Parameters

qualifier – the qualifier to check against

Returns

bool – True if this identifier was used, False otherwise

Raises

TypeError – if qualifier is None

has_operator(*operator: Operator*) → bool

Check if expression used a given operator.

Parameters

operator – the operator to check

Returns

bool – True if the given operator was used, False otherwise

Raises

TypeError – if operator is None

is_regex_flags_empty() → *bool*

Check if a regex flag set is empty.

Returns

bool – True if no regex flags where set, False otherwise

has_regex_flag(flag: *RegexFlag*) → *bool*

Check if a regex flag is set.

Parameters

flag – the flag to be checked

Returns

bool – True if the flag is set, False otherwise

Raises

TypeError – if flag is None

accept(visitor: *QueryVisitor*) → *None*

parent: *QueryNode* | *None*

The parent node of this node.

None if this is the root node.

class fcsql.parser.ExpressionWildcard(children: *List[QueryNode]* | *None* = *None*, child: *QueryNode* | *None* = *None*)

A FCS-QL expression tree WILDCARD expression node.

[Constructor]

Parameters

- **node_type** – the type of the node
- **children** – the children of this node or None. Defaults to None.
- **child** – the child of this node or None. Defaults to None.

accept(visitor: *QueryVisitor*) → *None*

parent: *QueryNode* | *None*

The parent node of this node.

None if this is the root node.

class fcsql.parser.ExpressionGroup(child: *QueryNode*)

A FCS-QL expression tree GROUP expression node.

[Constructor]

Parameters

child – the group content

accept(visitor: *QueryVisitor*) → *None*

parent: *QueryNode* | *None*

The parent node of this node.

None if this is the root node.

class fcsql.parser.**ExpressionNot**(child: QueryNode)

A FCS-QL expression tree NOT expression node.

[Constructor]

Parameters

child – the child expression

accept(visitor: QueryVisitor) → None

parent: QueryNode | None

The parent node of this node.

None if this is the root node.

class fcsql.parser.**ExpressionAnd**(children: List[QueryNode])

A FCS-QL expression tree AND expression node.

[Constructor]

Parameters

children – child elements covered by AND expression.

property operands: List[QueryNode]

Get the AND expression operands.

Returns

List[QueryNode] – a list of expressions

accept(visitor: QueryVisitor) → None

parent: QueryNode | None

The parent node of this node.

None if this is the root node.

class fcsql.parser.**ExpressionOr**(children: List[QueryNode])

A FCS-QL expression tree OR expression node.

[Constructor]

Parameters

children – child elements covered by OR expression.

property operands: List[QueryNode]

Get the OR expression operands.

Returns

List[QueryNode] – a list of expressions

accept(visitor: QueryVisitor) → None

parent: QueryNode | None

The parent node of this node.

None if this is the root node.

class fcsql.parser.**QueryDisjunction**(children: List[QueryNode])

A FCS-QL expression tree QR query.

[Constructor]

Parameters

children – the children

accept(*visitor*: *QueryVisitor*) → *None*

parent: *QueryNode* | *None*

The parent node of this node.

None if this is the root node.

class *fcsql.parser.QuerySequence*(*children*: *List[QueryNode]*)

A FCS-QL expression tree query sequence node.

[Constructor]

Parameters

children – the children for this node

accept(*visitor*: *QueryVisitor*) → *None*

parent: *QueryNode* | *None*

The parent node of this node.

None if this is the root node.

class *fcsql.parser.QueryWithWithin*(*query*: *QueryNode*, *within*: *QueryNode* | *None*)

FCS-QL expression tree QUERY-WITH-WITHIN node.

[Constructor]

Parameters

- **query** – the query node
- **within** – the within node

get_query() → *QueryNode*

Get the query clause.

Returns

QueryNode – the query clause

get_within() → *QueryNode* | *None*

Get the within clause (= search context)

Returns

QueryNode – the within clause

accept(*visitor*: *QueryVisitor*) → *None*

parent: *QueryNode* | *None*

The parent node of this node.

None if this is the root node.

class *fcsql.parser.QuerySegment*(*expression*: *QueryNode*, *min_occurs*: *int*, *max_occurs*: *int*)

A FCS-QL expression tree query segment node.

[Constructor]

Parameters

- **expression** – the expression

- **min_occurs** – the minimum occurrence
- **max_occurs** – the maximum occurrence

min_occurs

The minimum occurrence of this segment.

max_occurs

The maximum occurrence of this segment.

get_expression() → *QueryNode*

Get the expression for this segment.

Returns

QueryNode – the expression

accept(visitor: QueryVisitor) → None**parent:** *QueryNode* | None

The parent node of this node.

None if this is the root node.

class fcsql.parser.**QueryGroup**(child: *QueryNode*, min_occurs: *int*, max_occurs: *int*)

A FCS-QL expression tree GROUP query node.

[Constructor]

Parameters

- **child** – the child
- **min_occurs** – the minimum occurrence
- **max_occurs** – the maximum occurrence

min_occurs

The minimum occurrence of group content.

max_occurs

The maximum occurrence of group content.

get_content() → *QueryNode*

Get the group content.

Returns

QueryNode – the content of the GROUP query

accept(visitor: QueryVisitor) → None**parent:** *QueryNode* | None

The parent node of this node.

None if this is the root node.

class fcsql.parser.**SimpleWithin**(scope: *SimpleWithinScope*)

A FCS-QL expression tree SIMPLE WITHIN query node.

[Constructor]

Parameters

- **node_type** – the type of the node

- **children** – the children of this node or None. Defaults to None.
- **child** – the child of this node or None. Defaults to None.

scope

The simple within scope.

accept(*visitor*: [QueryVisitor](#)) → None

parent: [QueryNode](#) | None

The parent node of this node.

None if this is the root node.

`fcsql.parser.DEFAULT_UNICODE_NORMALIZATION_FORM = 'NFC'`

Default unicode normalization form.

See also: [unicodedata.normalize](#)

class `fcsql.parser.ErrorListener`(*query*: *str*)

syntaxError(*recognizer*, *offendingSymbol*, *line*, *column*, *msg*, *e*)

has_errors() → bool

exception `fcsql.parser.QueryParserException`

Query parser exception.

exception `fcsql.parser.ExpressionTreeBuilderException`

Error building expression tree.

class `fcsql.parser.ExpressionTreeBuilder`(*parser*: [QueryParser](#))

stack_Query_disjunction: [Deque](#)[int]

for *enterQuery_disjunction*/*exitQuery_disjunction*

stack_Query_sequence: [Deque](#)[int]

for *enterQuery_sequence*/*exitQuery_sequence*

stack_Expression_or: [Deque](#)[int]

for *enterExpression_or*/*exitExpression_or*

stack_Expression_and: [Deque](#)[int]

for *enterExpression_and*/*exitExpression_and*

enterQuery(*ctx*: [QueryContext](#))

exitQuery(*ctx*: [QueryContext](#))

enterMain_query(*ctx*: [Main_queryContext](#))

exitMain_query(*ctx*: [Main_queryContext](#))

enterQuery_disjunction(*ctx*: [Query_disjunctionContext](#))

exitQuery_disjunction(*ctx*: [Query_disjunctionContext](#))

enterQuery_sequence(*ctx*: [Query_sequenceContext](#))

exitQuery_sequence(*ctx*: [Query_sequenceContext](#))

```

enterQuery_group(ctx: Query_groupContext)
exitQuery_group(ctx: Query_groupContext)
enterQuery_simple(ctx: Query_simpleContext)
exitQuery_simple(ctx: Query_simpleContext)
enterQuery_implicit(ctx: Query_implicitContext)
exitQuery_implicit(ctx: Query_implicitContext)
enterQuery_segment(ctx: Query_segmentContext)
exitQuery_segment(ctx: Query_segmentContext)
enterExpression_basic(ctx: Expression_basicContext)
exitExpression_basic(ctx: Expression_basicContext)
enterExpression_not(ctx: Expression_notContext)
exitExpression_not(ctx: Expression_notContext)
enterExpression_group(ctx: Expression_groupContext)
exitExpression_group(ctx: Expression_groupContext)
enterExpression_or(ctx: Expression_orContext)
exitExpression_or(ctx: Expression_orContext)
enterExpression_and(ctx: Expression_andContext)
exitExpression_and(ctx: Expression_andContext)
enterAttribute(ctx: AttributeContext)
exitAttribute(ctx: AttributeContext)
enterRegexp(ctx: RegexpContext)
exitRegexp(ctx: RegexpContext)
enterWithin_part_simple(ctx: Within_part_simpleContext)
exitWithin_part_simple(ctx: Within_part_simpleContext)
static processRepetition(ctx: QualifierContext) → Tuple[int, int]
static processRepetitionRange(ctx: QuantifierContext) → Tuple[int, int]
static getChildIndex(ctx: ParserRuleContext, start: int, ttype: int) → int
static parseInt(val: str) → int
static stripQuotes(val: str) → str
static unescapeString(val: str) → str
static unescapeUnicode(val: str, i: int, size: int) → str

```

```
static parseHexChar(val: str) → int
```

```
class fcsql.parser.QueryParser(default_identifier: str = 'text', default_operator: Operator =  
                                Operator.EQUALS, unicode_normalization_form: str | None = 'NFC')
```

A FCS-QL query parser that produces FCS-QL expression trees.

[Constructor]

Parameters

- **default_identifier** – the default identifier to be used for simple expressions. Defaults to *DEFAULT_IDENTIFIER*.
- **default_operator** – the default operator. Defaults to *DEFAULT_OPERATOR*.
- **unicode_normalization_form** – the Unicode normalization form to be used or None to not perform normalization. Defaults to *DEFAULT_UNICODE_NORMALIZATION_FORM*.

```
parse(query: str) → QueryNode
```

Parse query.

Parameters

query – the raw FCS-QL query

Raises

QueryParserException – if an error occurred

Returns

QueryNode – a FCS-QL expression tree

2.3 ANTLR generated parser

See general information about auto-generated parsers at the [ANTLR home page](#) and for python at [antlr4 github docs \(python-target\)](#)

2.3.1 fcsql.FCSLexer

```
class fcsql.FCSLexer.FCSLexer
```

2.3.2 fcsql.FCSParser

```
class fcsql.FCSParser.FCSParser
```

```
__init__(self, input: TokenStream, output: TextIO = sys.stdout)
```

Used like this:

```
query: str = "some query"  
input_stream = antlr4.InputStream(query)  
lexer = fcsql.FCSLexer(input_stream)  
stream = antlr4.CommonTokenStream(lexer)  
parser = fcsql.FCSParser(stream)
```

Parameters

input (*TokenStream*) – The person sending the message

query(*self*) → FCSParser.QueryContext

Start the parsing process for the *query* rule (see BNF)

2.3.3 fcsql.FCSParserListener

class fcsql.FCSParserListener.FCSParserListener

INDICES AND TABLES

- `genindex`
- `modindex`

PYTHON MODULE INDEX

f

`fcsql`, [5](#)

`fcsql.FCSLexer`, [16](#)

`fcsql.FCSParser`, [16](#)

`fcsql.FCSParserListener`, [17](#)

`fcsql.parser`, [5](#)

Symbols

`__init__()` (*fcsql.FCSParser.FCSParser* method), 16

A

`accept()` (*fcsql.parser.Expression* method), 10
`accept()` (*fcsql.parser.ExpressionAnd* method), 11
`accept()` (*fcsql.parser.ExpressionGroup* method), 10
`accept()` (*fcsql.parser.ExpressionNot* method), 11
`accept()` (*fcsql.parser.ExpressionOr* method), 11
`accept()` (*fcsql.parser.ExpressionWildcard* method), 10
`accept()` (*fcsql.parser.QueryDisjunction* method), 12
`accept()` (*fcsql.parser.QueryGroup* method), 13
`accept()` (*fcsql.parser.QueryNode* method), 8
`accept()` (*fcsql.parser.QuerySegment* method), 13
`accept()` (*fcsql.parser.QuerySequence* method), 12
`accept()` (*fcsql.parser.QueryWithWithin* method), 12
`accept()` (*fcsql.parser.SimpleWithin* method), 14
`antlr_parse()` (in module *fcsql*), 5

C

`CASE_INSENSITIVE` (*fcsql.parser.RegexFlag* attribute), 6
`CASE_SENSITIVE` (*fcsql.parser.RegexFlag* attribute), 6
`char` (*fcsql.parser.RegexFlag* attribute), 6
`child_count` (*fcsql.parser.QueryNode* property), 8
`children` (*fcsql.parser.QueryNode* attribute), 7

D

`DEFAULT_UNICODE_NORMALIZATION_FORM` (in module *fcsql.parser*), 14

E

`enterAttribute()` (*fcsql.parser.ExpressionTreeBuilder* method), 15
`enterExpression_and()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`enterExpression_basic()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`enterExpression_group()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15

`enterExpression_not()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`enterExpression_or()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`enterMain_query()` (*fc-sql.parser.ExpressionTreeBuilder* method), 14
`enterQuery()` (*fcsql.parser.ExpressionTreeBuilder* method), 14
`enterQuery_disjunction()` (*fc-sql.parser.ExpressionTreeBuilder* method), 14
`enterQuery_group()` (*fc-sql.parser.ExpressionTreeBuilder* method), 14
`enterQuery_implicit()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`enterQuery_segment()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`enterQuery_sequence()` (*fc-sql.parser.ExpressionTreeBuilder* method), 14
`enterQuery_simple()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`enterRegexp()` (*fcsql.parser.ExpressionTreeBuilder* method), 15
`enterWithin_part_simple()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15
`EQUALS` (*fcsql.parser.Operator* attribute), 6
`ErrorListener` (class in *fcsql.parser*), 14
`ExceptionThrowingErrorListener` (class in *fcsql*), 5
`exitAttribute()` (*fcsql.parser.ExpressionTreeBuilder* method), 15
`exitExpression_and()` (*fc-sql.parser.ExpressionTreeBuilder* method), 15

<code>exitExpression_basic()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>ExpressionOr</code> (<i>class in fcsql.parser</i>), 11
<code>exitExpression_group()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>ExpressionTreeBuilder</code> (<i>class in fcsql.parser</i>), 14
<code>exitExpression_not()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>ExpressionTreeBuilderException</code> , 14
<code>exitExpression_or()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>ExpressionWildcard</code> (<i>class in fcsql.parser</i>), 10
<code>exitMain_query()</code>	(<i>fcsql.parser.ExpressionTreeBuilder</i> method), 14	F
<code>exitQuery()</code>	(<i>fcsql.parser.ExpressionTreeBuilder</i> method), 14	<code>FCSLexer</code> (<i>class in fcsql.FCSLexer</i>), 16
<code>exitQuery_disjunction()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 14	<code>FCSParser</code> (<i>class in fcsql.FCSParser</i>), 16
<code>exitQuery_group()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>FCSParserListener</code> (<i>class in fcsql.FCSParserListener</i>), 17
<code>exitQuery_implicit()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>fcsql</code> module, 5
<code>exitQuery_segment()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>fcsql.FCSLexer</code> module, 16
<code>exitQuery_sequence()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 14	<code>fcsql.FCSParser</code> module, 16
<code>exitQuery_simple()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	<code>fcsql.FCSParserListener</code> module, 17
<code>exitRegexp()</code>	(<i>fcsql.parser.ExpressionTreeBuilder</i> method), 15	<code>fcsql.parser</code> module, 5
<code>exitWithin_part_simple()</code>	(<i>fc-sql.parser.ExpressionTreeBuilder</i> method), 15	G
<code>Expression</code> (<i>class in fcsql.parser</i>), 8		<code>get_child()</code> (<i>fcsql.parser.QueryNode</i> method), 8
<code>EXPRESSION</code> (<i>fcsql.parser.QueryNodeType</i> attribute), 6		<code>get_content()</code> (<i>fcsql.parser.QueryGroup</i> method), 13
<code>EXPRESSION_AND</code> (<i>fcsql.parser.QueryNodeType</i> attribute), 6		<code>get_expression()</code> (<i>fcsql.parser.QuerySegment</i> method), 13
<code>EXPRESSION_GROUP</code> (<i>fcsql.parser.QueryNodeType</i> attribute), 6		<code>get_first_child()</code> (<i>fcsql.parser.QueryNode</i> method), 8
<code>EXPRESSION_NOT</code> (<i>fcsql.parser.QueryNodeType</i> attribute), 6		<code>get_last_child()</code> (<i>fcsql.parser.QueryNode</i> method), 8
<code>EXPRESSION_OR</code> (<i>fcsql.parser.QueryNodeType</i> attribute), 6		<code>get_query()</code> (<i>fcsql.parser.QueryWithWithin</i> method), 12
<code>EXPRESSION_WILDCARD</code> (<i>fcsql.parser.QueryNodeType</i> attribute), 6		<code>get_within()</code> (<i>fcsql.parser.QueryWithWithin</i> method), 12
<code>ExpressionAnd</code> (<i>class in fcsql.parser</i>), 11		<code>getChildIndex()</code> (<i>fcsql.parser.ExpressionTreeBuilder</i> static method), 15
<code>ExpressionGroup</code> (<i>class in fcsql.parser</i>), 10		H
<code>ExpressionNot</code> (<i>class in fcsql.parser</i>), 10		<code>has_errors()</code> (<i>fcsql.parser.ErrorListener</i> method), 14
		<code>has_layer_identifier()</code> (<i>fcsql.parser.Expression</i> method), 9
		<code>has_layer_qualifier()</code> (<i>fcsql.parser.Expression</i> method), 9
		<code>has_node_type()</code> (<i>fcsql.parser.QueryNode</i> method), 7
		<code>has_operator()</code> (<i>fcsql.parser.Expression</i> method), 9
		<code>has_regex_flag()</code> (<i>fcsql.parser.Expression</i> method), 10
		I
		<code>identifier</code> (<i>fcsql.parser.Expression</i> attribute), 9
		<code>IGNORE_DIACRITICS</code> (<i>fcsql.parser.RegexFlag</i> attribute), 6
		<code>is_layer_qualifier_empty()</code> (<i>fc-sql.parser.Expression</i> method), 9

- `is_regex_flags_empty()` (*fcsql.parser.Expression* method), 9
- ## L
- `LITERAL_MATCHING` (*fcsql.parser.RegexFlag* attribute), 6

M

`max_occurs` (*fcsql.parser.QueryGroup* attribute), 13
`max_occurs` (*fcsql.parser.QuerySegment* attribute), 13
`min_occurs` (*fcsql.parser.QueryGroup* attribute), 13
`min_occurs` (*fcsql.parser.QuerySegment* attribute), 13
module
 fcsql, 5
 fcsql.FCSLexer, 16
 fcsql.FCSParser, 16
 fcsql.FCSParserListener, 17
 fcsql.parser, 5

N

`node_type` (*fcsql.parser.QueryNode* attribute), 7
`NOT_EQUALS` (*fcsql.parser.Operator* attribute), 6

O

`OCCURS_UNBOUNDED` (in module *fcsql.parser*), 5
`operands` (*fcsql.parser.ExpressionAnd* property), 11
`operands` (*fcsql.parser.ExpressionOr* property), 11
`Operator` (class in *fcsql.parser*), 6
`operator` (*fcsql.parser.Expression* attribute), 9

P

`PARAGRAPH` (*fcsql.parser.SimpleWithinScope* attribute), 6
`parent` (*fcsql.parser.Expression* attribute), 10
`parent` (*fcsql.parser.ExpressionAnd* attribute), 11
`parent` (*fcsql.parser.ExpressionGroup* attribute), 10
`parent` (*fcsql.parser.ExpressionNot* attribute), 11
`parent` (*fcsql.parser.ExpressionOr* attribute), 11
`parent` (*fcsql.parser.ExpressionWildcard* attribute), 10
`parent` (*fcsql.parser.QueryDisjunction* attribute), 12
`parent` (*fcsql.parser.QueryGroup* attribute), 13
`parent` (*fcsql.parser.QueryNode* attribute), 7
`parent` (*fcsql.parser.QuerySegment* attribute), 13
`parent` (*fcsql.parser.QuerySequence* attribute), 12
`parent` (*fcsql.parser.QueryWithWithin* attribute), 12
`parent` (*fcsql.parser.SimpleWithin* attribute), 14
`parse()` (*fcsql.parser.QueryParser* method), 16
`parse()` (in module *fcsql*), 5
`parseHexChar()` (*fcsql.parser.ExpressionTreeBuilder* static method), 15
`parseInt()` (*fcsql.parser.ExpressionTreeBuilder* static method), 15
`processRepetition()` (*fcsql.parser.ExpressionTreeBuilder* static method), 15

`processRepetitionRange()` (*fcsql.parser.ExpressionTreeBuilder* static method), 15

Q

`qualifier` (*fcsql.parser.Expression* attribute), 9
`query()` (*fcsql.FCSParser.FCSParser* method), 17
`QUERY_DISJUNCTION` (*fcsql.parser.QueryNodeType* attribute), 5
`QUERY_GROUP` (*fcsql.parser.QueryNodeType* attribute), 5
`QUERY_SEGMENT` (*fcsql.parser.QueryNodeType* attribute), 5
`QUERY_SEQUENCE` (*fcsql.parser.QueryNodeType* attribute), 5
`QUERY_WITH_WITHIN` (*fcsql.parser.QueryNodeType* attribute), 5
`QueryDisjunction` (class in *fcsql.parser*), 11
`QueryGroup` (class in *fcsql.parser*), 13
`QueryNode` (class in *fcsql.parser*), 7
`QueryNodeType` (class in *fcsql.parser*), 5
`QueryParser` (class in *fcsql.parser*), 16
`QueryParserException`, 14
`QuerySegment` (class in *fcsql.parser*), 12
`QuerySequence` (class in *fcsql.parser*), 12
`QueryVisitor` (class in *fcsql.parser*), 7
`QueryWithWithin` (class in *fcsql.parser*), 12

R

`regex` (*fcsql.parser.Expression* attribute), 9
`regex_flags` (*fcsql.parser.Expression* attribute), 9
`RegexFlag` (class in *fcsql.parser*), 6

S

`scope` (*fcsql.parser.SimpleWithin* attribute), 14
`SENTENCE` (*fcsql.parser.SimpleWithinScope* attribute), 6
`SESSION` (*fcsql.parser.SimpleWithinScope* attribute), 7
`SIMPLE_WITHIN` (*fcsql.parser.QueryNodeType* attribute), 6
`SimpleWithin` (class in *fcsql.parser*), 13
`SimpleWithinScope` (class in *fcsql.parser*), 6
`stack_expression_and` (*fcsql.parser.ExpressionTreeBuilder* attribute), 14
`stack_expression_or` (*fcsql.parser.ExpressionTreeBuilder* attribute), 14
`stack_query_disjunction` (*fcsql.parser.ExpressionTreeBuilder* attribute), 14
`stack_query_sequence` (*fcsql.parser.ExpressionTreeBuilder* attribute), 14
`stripQuotes()` (*fcsql.parser.ExpressionTreeBuilder* static method), 15

SyntaxError, 5

syntaxError() (*fcsql.ExceptionThrowingErrorListener*
method), 5

syntaxError() (*fcsql.parser.ErrorListener* method), 14

T

TEXT (*fcsql.parser.SimpleWithinScope* attribute), 7

TURN (*fcsql.parser.SimpleWithinScope* attribute), 7

U

unescapeString() (*fcsql.parser.ExpressionTreeBuilder*
static method), 15

unescapeUnicode() (fcs-
sql.parser.ExpressionTreeBuilder static
method), 15

UTTERANCE (*fcsql.parser.SimpleWithinScope* attribute), 6

V

visit() (*fcsql.parser.QueryVisitor* method), 7