FCS SRU Server

Release 0.1

Erik Körner

Nov 01, 2023

CONTENTS

1 FCS SRU Server **1**
- 1.1 Summary . . . 1
- 1.2 Installation . . . 2
- 1.3 Build source/binary distribution . . . 2
- 1.4 Development . . . 2
- 1.5 Build documentation . . . 3
- 1.6 See also . . . 3

2 Reference **5**
- 2.1 clarin.sru.constants . . . 5
- 2.2 clarin.sru.diagnostic . . . 11
- 2.3 clarin.sru.exception . . . 12
- 2.4 clarin.sru.queryparser . . . 12
- 2.5 clarin.sru.server.auth . . . 15
- 2.6 clarin.sru.server.config . . . 16
- 2.7 clarin.sru.server.request . . . 22
- 2.8 clarin.sru.server.result . . . 32
- 2.9 clarin.sru.server.server . . . 37
- 2.10 clarin.sru.server.wsgi . . . 39
- 2.11 clarin.sru.xml.writer . . . 40

3 Indices and tables **45**

Python Module Index **47**

Index **49**

CHAPTER

ONE

FCS SRU SERVER

- Based on Java implementation *git commit:* `0091fca0a4add134c478beed422dd1399a5364e3`
- Differences:
  - a bit more pythonic (naming, interfaces, enums etc.)
  - no auth stuff yet
  - WIP output buffering, server framework might not allow this, so no streaming and everything is in memory until sent
  - server framework choice (wsgi, asgi), for now `werkzeug`
  - TODO: refactoring to allow async variants for streaming responses (large resources), e.g. with `starlette`

1.1 Summary

This package implements the server-side part of the SRU/CQL protocol (SRU/S) and conforms to SRU version 1.1 and 1.2. SRU version 2.0 is mostly implemented but might be missing some more obscure features. The library will handle most of the protocol related tasks for you and you'll only need to implement a few classes to connect you search engine. However, the library will not save you from doing your SRU/CQL homework (i.e. you'll need to have at least some understanding of the protocol and adhere to the protocol semantics). Furthermore, you need to have at least some basic understanding of Python web application development (wsgi in particular) to use this library.

More Information about SRU/CQL: http://www.loc.gov/standards/sru/

The implementation is designed to make very minimal assumptions about the environment it's deployed in. For interfacing with your search engine, you need to implement the `SRUSearchEngine` interface. At minimum, you'll need to implement at least the `search()` method. Please check the Python API documentation for further details about this interface. The `SRUServer` implements the SRU protocol and uses your supplied search engine implementation to talk to your search engine. The SRUServer is configured using a `SRUServerConfig` instance. The `SRUServerConfig` reads an XML document, which contains the (static) server configuration. It must conform to the `sru-server-config.xsd` schema in the *`src/clarin/sru/xml/`* directory.

1.2 Installation

```
# from github/source
python3 -m pip install 'fcs-sru-server @ git+https://github.com/Querela/fcs-sru-server-
↪python.git'

# (locally) built package
python3 -m pip install dist/fcs_sru_server-<version>-py2.py3-none-any.whl
# or
python3 -m pip install dist/fcs-sru-server-<version>.tar.gz

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

In `setup.cfg`:

```
[options]
install_requires =
    fcs-sru-server @ git+https://github.com/Querela/fcs-sru-server-python.git
```

1.3 Build source/binary distribution

```
python3 -m pip install build
python3 -m build
```

1.4 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 .

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

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

(continues on next page)

(continued from previous page)

```
# 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
```

1.5 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
```

1.6 See also

- clarin-eric/fcs-sru-server
- clarin-eric/fcs-sru-client

CHAPTER

TWO

REFERENCE

2.1 clarin.sru.constants

class clarin.sru.constants.**SRUOperation**(*value*)

Bases: str, Enum

SRU operation

EXPLAIN = 'explain'

A explain operation

SEARCH_RETRIEVE = 'searchRetrieve'

A searchRetrieve operation

SCAN = 'scan'

A scan operation

class clarin.sru.constants.**SRUQueryType**(*value*)

Bases: str, Enum

An enumeration.

CQL = 'cql'

shorthand queryType identifier for CQL

SEARCH_TERMS = 'searchTerms'

class clarin.sru.constants.**SRURecordPacking**(*value*)

Bases: str, Enum

SRU 2.0 record packing.

PACKED = 'packed'

The client requests that the server should supply records strictly according to the requested schema.

UNPACKED = 'unpacked'

The server is free to allow the location of application data to vary within the record.

class clarin.sru.constants.**SRURecordXmlEscaping**(*value*)

Bases: str, Enum

SRU Record XML escaping (or record packing in SRU 1.2).

XML = 'xml'

XML record packing

STRING = 'string'

String record packing

class clarin.sru.constants.**SRURenderBy**(*value*)

Bases: str, Enum

SRU Record XML escaping.

CLIENT = 'client'

The client requests that the server simply return this URL in the response, in the href attribute of the xml-stylesheet processing instruction before the response xml.

SERVER = 'server'

The client requests that the server format the response according to the specified stylesheet, assuming the default SRU response schema as input to the stylesheet.

class clarin.sru.constants.**SRUResultCountPrecision**(*value*)

Bases: str, Enum

(SRU 2.0) Indicate the accuracy of the result count reported by total number of records that matched the query.

EXACT = 'exact'

The server guarantees that the reported number of records is accurate.

UNKNOWN = 'unknown'

The server has no idea what the result count is, and does not want to venture an estimate.

ESTIMATE = 'estimate'

The server does not know the result set count, but offers an estimate.

MAXIMUM = 'maximum'

The value supplied is an estimate of the maximum possible count that the result set will attain.

MINIMUM = 'minimum'

The server does not know the result count but guarantees that it is at least this large.

CURRENT = 'current'

The value supplied is an estimate of the count at the time the response was sent, however the result set may continue to grow.

class clarin.sru.constants.**SRUVersion**(*value*)

Bases: str, Enum

SRU version

major: int

minor: int

property version_number: int

property version_string: str

VERSION_1_1 = '1.1'

VERSION_1_2 = '1.2'

VERSION_2_0 = '2.0'

class clarin.sru.constants.**SRUDiagnostics**(*value*)

Bases: str, Enum

Constants for SRU diagnostics

See also:

- SRU Diagnostics: http://www.loc.gov/standards/sru/diagnostics/
- SRU Diagnostics List: http://www.loc.gov/standards/sru/diagnostics/diagnosticsList.html

nr: int

category: str

description: str

GENERAL_SYSTEM_ERROR = 'info:srw/diagnostic/1/1'

SYSTEM_TEMPORARILY_UNAVAILABLE = 'info:srw/diagnostic/1/2'

AUTHENTICATION_ERROR = 'info:srw/diagnostic/1/3'

UNSUPPORTED_OPERATION = 'info:srw/diagnostic/1/4'

UNSUPPORTED_VERSION = 'info:srw/diagnostic/1/5'

UNSUPPORTED_PARAMETER_VALUE = 'info:srw/diagnostic/1/6'

MANDATORY_PARAMETER_NOT_SUPPLIED = 'info:srw/diagnostic/1/7'

UNSUPPORTED_PARAMETER = 'info:srw/diagnostic/1/8'

DATABASE_DOES_NOT_EXIST = 'info:srw/diagnostic/1/235'

QUERY_SYNTAX_ERROR = 'info:srw/diagnostic/1/10'

TOO_MANY_CHARACTERS_IN_QUERY = 'info:srw/diagnostic/1/12'

INVALID_OR_UNSUPPORTED_USE_OF_PARENTHESES = 'info:srw/diagnostic/1/13'

INVALID_OR_UNSUPPORTED_USE_OF_QUOTES = 'info:srw/diagnostic/1/14'

UNSUPPORTED_CONTEXT_SET = 'info:srw/diagnostic/1/15'

UNSUPPORTED_INDEX = 'info:srw/diagnostic/1/16'

UNSUPPORTED_COMBINATION_OF_INDEXES = 'info:srw/diagnostic/1/18'

UNSUPPORTED_RELATION = 'info:srw/diagnostic/1/19'

UNSUPPORTED_RELATION_MODIFIER = 'info:srw/diagnostic/1/20'

UNSUPPORTED_COMBINATION_OF_RELATION_MODIFERS = 'info:srw/diagnostic/1/21'

UNSUPPORTED_COMBINATION_OF_RELATION_AND_INDEX = 'info:srw/diagnostic/1/22'

TOO_MANY_CHARACTERS_IN_TERM = 'info:srw/diagnostic/1/23'

UNSUPPORTED_COMBINATION_OF_RELATION_AND_TERM = 'info:srw/diagnostic/1/24'

```
NON_SPECIAL_CHARACTER_ESCAPED_IN_TERM = 'info:srw/diagnostic/1/26'

EMPTY_TERM_UNSUPPORTED = 'info:srw/diagnostic/1/27'

MASKING_CHARACTER_NOT_SUPPORTED = 'info:srw/diagnostic/1/28'

MASKED_WORDS_TOO_SHORT = 'info:srw/diagnostic/1/29'

TOO_MANY_MASKING_CHARACTERS_IN_TERM = 'info:srw/diagnostic/1/30'

ANCHORING_CHARACTER_NOT_SUPPORTED = 'info:srw/diagnostic/1/31'

ANCHORING_CHARACTER_IN_UNSUPPORTED_POSITION = 'info:srw/diagnostic/1/32'

COMBINATION_OF_PROXIMITY_ADJACENCY_AND_MASKING_CHARACTERS_NOT_SUPPORTED =
'info:srw/diagnostic/1/33'

COMBINATION_OF_PROXIMITY_ADJACENCY_AND_ANCHORING_CHARACTERS_NOT_SUPPORTED =
'info:srw/diagnostic/1/34'

TERM_CONTAINS_ONLY_STOPWORDS = 'info:srw/diagnostic/1/35'

TERM_IN_INVALID_FORMAT_FOR_INDEX_OR_RELATION = 'info:srw/diagnostic/1/36'

UNSUPPORTED_BOOLEAN_OPERATOR = 'info:srw/diagnostic/1/37'

TOO_MANY_BOOLEAN_OPERATORS_IN_QUERY = 'info:srw/diagnostic/1/38'

PROXIMITY_NOT_SUPPORTED = 'info:srw/diagnostic/1/39'

UNSUPPORTED_PROXIMITY_RELATION = 'info:srw/diagnostic/1/40'

UNSUPPORTED_PROXIMITY_DISTANCE = 'info:srw/diagnostic/1/41'

UNSUPPORTED_PROXIMITY_UNIT = 'info:srw/diagnostic/1/42'

UNSUPPORTED_PROXIMITY_ORDERING = 'info:srw/diagnostic/1/43'

UNSUPPORTED_COMBINATION_OF_PROXIMITY_MODIFIERS = 'info:srw/diagnostic/1/44'

UNSUPPORTED_BOOLEAN_MODIFIER = 'info:srw/diagnostic/1/46'

CANNOT_PROCESS_QUERY_REASON_UNKNOWN = 'info:srw/diagnostic/1/47'

QUERY_FEATURE_UNSUPPORTED = 'info:srw/diagnostic/1/48'

MASKING_CHARACTER_IN_UNSUPPORTED_POSITION = 'info:srw/diagnostic/1/49'

RESULT_SETS_NOT_SUPPORTED = 'info:srw/diagnostic/1/50'

RESULT_SET_DOES_NOT_EXIST = 'info:srw/diagnostic/1/51'

RESULT_SET_TEMPORARILY_UNAVAILABLE = 'info:srw/diagnostic/1/52'

RESULT_SETS_ONLY_SUPPORTED_FOR_RETRIEVAL = 'info:srw/diagnostic/1/53'

COMBINATION_OF_RESULT_SETS_WITH_SEARCH_TERMS_NOT_SUPPORTED =
'info:srw/diagnostic/1/55'
```

```
RESULT_SET_CREATED_WITH_UNPREDICTABLE_PARTIAL_RESULTS_AVAILABLE =
'info:srw/diagnostic/1/58'

RESULT_SET_CREATED_WITH_VALID_PARTIAL_RESULTS_AVAILABLE = 'info:srw/diagnostic/1/59'

RESULT_SET_NOT_CREATED_TOO_MANY_MATCHING_RECORDS = 'info:srw/diagnostic/1/60'

FIRST_RECORD_POSITION_OUT_OF_RANGE = 'info:srw/diagnostic/1/61'

RECORD_TEMPORARILY_UNAVAILABLE = 'info:srw/diagnostic/1/64'

RECORD_DOES_NOT_EXIST = 'info:srw/diagnostic/1/65'

UNKNOWN_SCHEMA_FOR_RETRIEVAL = 'info:srw/diagnostic/1/66'

RECORD_NOT_AVAILABLE_IN_THIS_SCHEMA = 'info:srw/diagnostic/1/67'

NOT_AUTHORISED_TO_SEND_RECORD = 'info:srw/diagnostic/1/68'

NOT_AUTHORISED_TO_SEND_RECORD_IN_THIS_SCHEMA = 'info:srw/diagnostic/1/69'

RECORD_TOO_LARGE_TO_SEND = 'info:srw/diagnostic/1/70'

UNSUPPORTED_XML_ESCAPING_VALUE = 'info:srw/diagnostic/1/71'

XPATH_RETRIEVAL_UNSUPPORTED = 'info:srw/diagnostic/1/72'

XPATH_EXPRESSION_CONTAINS_UNSUPPORTED_FEATURE = 'info:srw/diagnostic/1/73'

UNABLE_TO_EVALUATE_XPATH_EXPRESSION = 'info:srw/diagnostic/1/74'

SORT_NOT_SUPPORTED = 'info:srw/diagnostic/1/80'

UNSUPPORTED_SORT_SEQUENCE = 'info:srw/diagnostic/1/82'

TOO_MANY_RECORDS_TO_SORT = 'info:srw/diagnostic/1/83'

TOO_MANY_SORT_KEYS_TO_SORT = 'info:srw/diagnostic/1/84'

CANNOT_SORT_INCOMPATIBLE_RECORD_FORMATS = 'info:srw/diagnostic/1/86'

UNSUPPORTED_SCHEMA_FOR_SORT = 'info:srw/diagnostic/1/87'

UNSUPPORTED_PATH_FOR_SORT = 'info:srw/diagnostic/1/88'

PATH_UNSUPPORTED_FOR_SCHEMA = 'info:srw/diagnostic/1/89'

UNSUPPORTED_DIRECTION = 'info:srw/diagnostic/1/90'

UNSUPPORTED_CASE = 'info:srw/diagnostic/1/91'

UNSUPPORTED_MISSING_VALUE_ACTION = 'info:srw/diagnostic/1/92'

SORT_ENDED_DUE_TO_MISSING_VALUE = 'info:srw/diagnostic/1/93'

SORT_SPEC_INCLUDED_BOTH_IN_QUERY_AND_PROTOCOL_QUERY_PREVAILS =
'info:srw/diagnostic/1/94'

SORT_SPEC_INCLUDED_BOTH_IN_QUERY_AND_PROTOCOL_PROTOCOL_PREVAILS =
'info:srw/diagnostic/1/95'
```

SORT_SPEC_INCLUDED_BOTH_IN_QUERY_AND_PROTOCOL_ERROR = 'info:srw/diagnostic/1/96'

STYLESHEETS_NOT_SUPPORTED = 'info:srw/diagnostic/1/110'

UNSUPPORTED_STYLESHEET = 'info:srw/diagnostic/1/111'

RESPONSE_POSITION_OUT_OF_RANGE = 'info:srw/diagnostic/1/120'

TOO_MANY_TERMS_REQUESTED = 'info:srw/diagnostic/1/121'

classmethod get_by_uri(*uri: str*) → *SRUDiagnostics* | None

class clarin.sru.constants.SRUParam(*value*)

Bases: str, Enum

An enumeration.

OPERATION = 'operation'

VERSION = 'version'

STYLESHEET = 'stylesheet'

RENDER_BY = 'renderedBy'

HTTP_ACCEPT = 'httpAccept'

RESPONSE_TYPE = 'responseType'

QUERY = 'query'

QUERY_TYPE = 'queryType'

START_RECORD = 'startRecord'

MAXIMUM_RECORDS = 'maximumRecords'

RECORD_XML_ESCAPING = 'recordXMLEscaping'

RECORD_PACKING = 'recordPacking'

RECORD_SCHEMA = 'recordSchema'

RECORD_XPATH = 'recordXPath'

RESULT_SET_TTL = 'resultSetTTL'

SORT_KEYS = 'sortKeys'

SCAN_CLAUSE = 'scanClause'

RESPONSE_POSITION = 'responsePosition'

MAXIMUM_TERMS = 'maximumTerms'

X_UNLIMITED_RESULTSET = 'x-unlimited-resultset'

X_UNLIMITED_TERMLIST = 'x-unlimited-termlist'

X_INDENT_RESPONSE = 'x-indent-response'

class clarin.sru.constants.**SRUParamValue**(*value*)

Bases: str, Enum

An enumeration.

OP_EXPLAIN = 'explain'

OP_SCAN = 'scan'

OP_SEARCH_RETRIEVE = 'searchRetrieve'

VERSION_1_1 = '1.1'

VERSION_1_2 = '1.2'

RECORD_XML_ESCAPING_XML = 'xml'

RECORD_XML_ESCAPING_STRING = 'string'

RECORD_PACKING_PACKED = 'packed'

RECORD_PACKING_UNPACKED = 'unpacked'

RENDER_BY_CLIENT = 'client'

RENDER_BY_SERVER = 'server'

2.2 clarin.sru.diagnostic

class clarin.sru.diagnostic.**SRUDiagnostic**(*uri: str*, *details: str | None = None*, *message: str | None = None*)

Bases: object

Class to hold a SRU diagnostic.

See also:

- SRU Diagnostics: http://www.loc.gov/standards/sru/diagnostics/
- SRU Diagnostics List: http://www.loc.gov/standards/sru/diagnostics/diagnosticsList.html

uri: str

Diagnostic's identifying URI.

details: str | None = None

Supplementary information available, often in a format specified by the diagnostic or None.

message: str | None = None

Human readable message to display to the end user or None.

static get_default_error_message(*uri: str*)

class clarin.sru.diagnostic.**SRUDiagnosticList**

Bases: object

Container for non surrogate diagnostics for the request. The will be put in the diagnostics part of the response.

abstract add_diagnostic(*uri: str*, *details: str | None = None*, *message: str | None = None*) → None

Add a non surrogate diagnostic to the response.

Parameters

- **uri** – the diagnostic's identifying URI
- **details** – supplementary information available, often in a format specified by the diagnostic or `None`
- **message** – human readable message to display to the end user or `None`

2.3 clarin.sru.exception

exception clarin.sru.exception.**SRUException**(*uri: str*, *details: str | None = None*, *message: str | None = None*, **args*)

Bases: `Exception`

An exception raised, if something went wrong processing the request. For diagnostic codes, see constants in *SRUConstant*.

See also:

SRUConstant

get_diagnostic() → *SRUDiagnostic*

Create a SRU diagnostic from this exception.

exception clarin.sru.exception.**SRUConfigException**

Bases: `Exception`

An exception raised, if some error occurred with the SRUServer configuration.

2.4 clarin.sru.queryparser

class clarin.sru.queryparser.**SRUQuery**(*raw_query: str*, *parsed_query: _T*)

Bases: `ABC`, `Generic`[_T]

Holder class for a parsed query to be returned from a *SRUQueryParser*.

abstract property query_type: str

Get the short name for supported query, e.g. "cql".

property raw_query: str

Get the original query as a string.

property parsed_query: _T

Get the parsed query as an abstract syntax tree.

class clarin.sru.queryparser.**SRUQueryParser**(**args*, ***kwds*)

Bases: `ABC`, `Generic`[_T]

Interface for implementing pluggable query parsers.

Parameterized by 'abstract syntax tree (object) for parsed queries.'

abstract property query_type: `str`

Get the short name for supported query, e.g. "cql".

abstract supports_version(*version:* SRUVersion | *None*) → bool

Check if query is supported by a specific version of SRU/CQL.

property query_type_definition: `str | None`

The URI for the for the query type's definition.

abstract property query_parameter_names: `List[str]`

Get the list of query parameters.

abstract parse_query(*version:* SRUVersion, *parameters: Dict[str, str]*, *diagnostics:* SRUDiagnosticList) → *SRUQuery*[_T] | None

Parse a query into an abstract syntax tree.

Parameters

- **version** – the SRU version the request was made
- **parameters** – the request parameters containing the query
- **diagnostics** – a *SRUDiagnosticList* for storing fatal and non-fatal diagnostics

Returns

the parsed query or `None` if the query could not be parsed

class clarin.sru.queryparser.**SRUQueryParserRegistry**(*parsers: List[*SRUQueryParser*[Any]]*)

Bases: `object`

A registry to keep track of registered *SRUQueryParser* to be used by the *SRUServer*.

See also:

SRUQueryParser

property query_parsers: `List[SRUQueryParser[Any]]`

Get a list of all registered query parsers.

Returns

List[SRUQueryParser[Any]] –

a list of registered query parsers

find_query_parser(*query_type: str*) → *SRUQueryParser*[Any] | None

Find a query parser by query type.

Parameters

query_type – the query type to search for

Returns

SRUQueryParser[Any] –

the matching *SRUQueryParser* instance or `None` if no matching parser was found.

class Builder(*register_defaults: bool = True*)

Bases: `object`

Builder for creating *SRUQueryParserRegistry* instances.

[Constructor]

Parameters
register_defaults – if `True`, register SRU/CQL standard query parsers (queryType **cql** and **searchTerms**), otherwise do nothing. Defaults to True.

`register_defaults`() → *Builder*

Registers registers SRU/CQL standard query parsers (queryType **cql** and **searchTerms**).

`register`(*parser:* SRUQueryParser*[Any]*) → *Builder*

Register a new query parser
Parameters
parser (*SRUQueryParser[Any]*) – the query parser instance to be registered
Raises
SRUConfigException – if a query parser for the same query type was already registered

`build`() → *SRUQueryParserRegistry*

Create a configured *SRUQueryParserRegistry* instance from this builder.
Returns

SRUQueryParserRegistry –
a *SRUQueryParserRegistry*
instance

`class clarin.sru.queryparser.SearchTermsQuery`(*raw_query: str*, *parsed_query: _T*)

Bases: *SRUQuery*[`List`[`str`]]

`property query_type: str`

Get the short name for supported query, e.g. “cql”.

`class clarin.sru.queryparser.SearchTermsQueryParser`(**args*, ***kwds*)

Bases: *SRUQueryParser*[`List`[`str`]]

`property query_type: str`

Get the short name for supported query, e.g. “cql”.

`property query_parameter_names: List[str]`

Get the list of query parameters.

`supports_version`(*version:* SRUVersion | *None*) → bool

Check if query is supported by a specific version of SRU/CQL.

`parse_query`(*version:* SRUVersion, *parameters: Dict[str, str]*, *diagnostics:* SRUDiagnosticList) → *SRUQuery*[List[str]] | None

Parse a query into an abstract syntax tree.

Parameters

- **version** – the SRU version the request was made
- **parameters** – the request parameters containing the query
- **diagnostics** – a *SRUDiagnosticList* for storing fatal and non-fatal diagnostics

Returns
the parsed query or `None` if the query could not be parsed

`class clarin.sru.queryparser.CQLQuery`(*raw_query: str*, *parsed_query: _T*)

Bases: *SRUQuery*[`CQLQuery`]

`property query_type: str`

Get the short name for supported query, e.g. “cql”.

class clarin.sru.queryparser.**CQLQueryParser**(**args*, ***kwds*)

Bases: *SRUQueryParser*[CQLQuery]

Default query parser to parse CQL.

property query_type: str

Get the short name for supported query, e.g. "cql".

property query_parameter_names: List[str]

Get the list of query parameters.

supports_version(*version:* SRUVersion | *None*) → bool

Check if query is supported by a specific version of SRU/CQL.

parse_query(*version:* SRUVersion, *parameters:* *Dict[str, str]*, *diagnostics:* SRUDiagnosticList) → *SRUQuery*[CQLQuery] | None

Parse a query into an abstract syntax tree.

Parameters

- **version** – the SRU version the request was made
- **parameters** – the request parameters containing the query
- **diagnostics** – a *SRUDiagnosticList* for storing fatal and non-fatal diagnostics

Returns

the parsed query or None if the query could not be parsed

2.5 clarin.sru.server.auth

class clarin.sru.server.auth.**SRUAuthenticationInfo**

Bases: object

abstract property authentication_method: str

abstract property subject: str

class clarin.sru.server.auth.**SRUAuthenticationInfoProvider**

Bases: object

abstract get_AuthenticationInfo(*request:* *Request*) → *SRUAuthenticationInfo* | None

class clarin.sru.server.auth.**SRUAuthenticationInfoProviderFactory**

Bases: object

abstract create_SRUAuthenticationInfoProvider(*params:* *Dict[str, str]*) → *SRUAuthenticationInfoProvider* | None

Create a authentication info provider.

2.6 clarin.sru.server.config

class clarin.sru.server.config.**LegacyNamespaceMode**(*value*)

Bases: str, Enum

An enumeration.

LOC = 'loc'

OASIS = 'oasis'

class clarin.sru.server.config.**LocalizedString**(*value: str*, *lang: str*, *primary: bool = False*)

Bases: object

value: str

lang: str

primary: bool = False

class clarin.sru.server.config.**SRUServerConfigKey**(*value*)

Bases: str, Enum

An enumeration.

SRU_SUPPORTED_VERSION_MIN = 'eu.clarin.sru.server.sruSupportedVersionMin'

Parameter constant for setting the minimum supported SRU version for this SRU server. Must be smaller or equal to *SRU_SUPPORTED_VERSION_MAX*.

Valid values: "1.1", "1.2" or " 2.0" (without quotation marks)

SRU_SUPPORTED_VERSION_MAX = 'eu.clarin.sru.server.sruSupportedVersionMax'

Parameter constant for setting the maximum supported SRU version for this SRU server. Must be larger or equal to *SRU_SUPPORTED_VERSION_MIN*.

Valid values: "1.1", "1.2" or "2.0" (without quotation marks)

SRU_SUPPORTED_VERSION_DEFAULT = 'eu.clarin.sru.server.sruSupportedVersionDefault'

Parameter constant for setting the default SRU version for this SRU server, e.g. for an **Explain** request without explicit version.

Must not me less than *SRU_SUPPORTED_VERSION_MIN* or larger than *SRU_SUPPORTED_VERSION_MAX*. Defaults to *SRU_SUPPORTED_VERSION_MAX*.

Valid values: "1.1", "1.2" or "2.0" (without quotation marks)

SRU_LEGACY_NAMESPACE_MODE = 'eu.clarin.sru.server.legacyNamespaceMode'

Parameter constant for setting the namespace URIs for SRU 1.1 and SRU 1.2.

Valid values: "loc" for Library Of Congress URI or "oasis" for OASIS URIs (without quotation marks).

SRU_TRANSPORT = 'eu.clarin.sru.server.transport'

Parameter constant for configuring the transports for this SRU server.

Valid values: "http", "https" or "http https" (without quotation marks)

Used as part of the **Explain** response.

SRU_HOST = 'eu.clarin.sru.server.host'

Parameter constant for configuring the host of this SRU server.

Valid values: any fully qualified hostname, e.g. `sru.example.org`.

Used as part of the **Explain** response.

SRU_PORT = 'eu.clarin.sru.server.port'

Parameter constant for configuring the port number of this SRU server.

Valid values: number between 1 and 65535 (typically 80 or 8080)

Used as part of the **Explain** response.

SRU_DATABASE = 'eu.clarin.sru.server.database'

Parameter constant for configuring the database of this SRU server. This is usually the path component of the SRU servers URI.

Valid values: typically the path component if the SRU server URI.

Used as part of the **Explain** response.

SRU_NUMBER_OF_RECORDS = 'eu.clarin.sru.server.numberOfRecords'

Parameter constant for configuring the **default** number of records the SRU server will provide in the response to a **searchRetrieve** request if the client does not provide this value.

Valid values: a integer greater than 0 (default value is 100)

SRU_MAXIMUM_RECORDS = 'eu.clarin.sru.server.maximumRecords'

Parameter constant for configuring the **maximum** number of records the SRU server will support in the response to a **searchRetrieve** request. If a client requests more records, the number will be limited to this value.

Valid values: a integer greater than 0 (default value is 250)

SRU_NUMBER_OF_TERMS = 'eu.clarin.sru.server.numberOfTerms'

Parameter constant for configuring the **default** number of terms the SRU server will provide in the response to a **scan** request if the client does not provide this value.

Valid values: a integer greater than 0 (default value is 250)

SRU_MAXIMUM_TERMS = 'eu.clarin.sru.server.maximumTerms'

Parameter constant for configuring the **maximum** number of terms the SRU server will support in the response to a **scan** request. If a client requests more records, the number will be limited to this value.

Valid values: a integer greater than 0 (default value is 500)

SRU_ECHO_REQUESTS = 'eu.clarin.sru.server.echoRequests'

Parameter constant for configuring, if the SRU server will echo the request.

Valid values: `true` or `false`

SRU_INDENT_RESPONSE = 'eu.clarin.sru.server.indentResponse'

Parameter constant for configuring, if the SRU server pretty-print the XML response. Setting this parameter can be useful for manual debugging of the XML response, however it is **not recommended** for production setups.

Valid values: any integer greater or equal to `-1` (default) and less or equal to `8`

SRU_ALLOW_OVERRIDE_MAXIMUM_RECORDS = 'eu.clarin.sru.server.allowOverrideMaximumRecords'

Parameter constant for configuring, if the SRU server will allow the client to override the maximum number of records the server supports. This parameter is solely intended for debugging and setting it to `true` is **strongly** discouraged for production setups.

Valid values: `true` or `false` (default)

SRU_ALLOW_OVERRIDE_MAXIMUM_TERMS = 'eu.clarin.sru.server.allowOverrideMaximumTerms'

Parameter constant for configuring, if the SRU server will allow the client to override the maximum number of terms the server supports. This parameter is solely intended for debugging and setting it to `true` it is **strongly** discouraged for production setups.

Valid values: `true` or `false` (default)

SRU_ALLOW_OVERRIDE_INDENT_RESPONSE = 'eu.clarin.sru.server.allowOverrideIndentResponse'

Parameter constant for configuring, if the SRU server will allow the client to override the pretty-printing setting of the server. This parameter is solely intended for debugging and setting it to `true` it is **strongly** discouraged for production setups.

Valid values: `true` or `false` (default)

SRU_RESPONSE_BUFFER_SIZE = 'eu.clarin.sru.server.responseBufferSize'

Parameter constant for configuring the size of response buffer. The Servlet will buffer up to this amount of data before sending a response to the client. This value specifies the size of the buffer in bytes.

Valid values: any positive integer (default 65536)

class clarin.sru.server.config.**DatabaseInfo**(*title: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *description: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *author: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *extent: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *history: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *langUsage: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *restrictions: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *subjects: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *links: List[clarin.sru.server.config.LocalizedString] | NoneType = None*, *implementation: List[clarin.sru.server.config.LocalizedString] | NoneType = None*)

Bases: `object`

title: *List[LocalizedString] | None* = **None**

description: *List[LocalizedString] | None* = **None**

author: *List[LocalizedString] | None* = **None**

extent: *List[LocalizedString] | None* = **None**

history: List[*LocalizedString*] | None = None

langUsage: List[*LocalizedString*] | None = None

restrictions: List[*LocalizedString*] | None = None

subjects: List[*LocalizedString*] | None = None

links: List[*LocalizedString*] | None = None

implementation: List[*LocalizedString*] | None = None

class clarin.sru.server.config.**SchemaInfo**(*identifier: str*, *name: str*, *location: str*, *sort: bool*, *retrieve: bool*, *title: List[*clarin.sru.server.config.LocalizedString*]* | *NoneType = None*)

Bases: object

identifier: str

name: str

location: str

sort: bool

retrieve: bool

title: List[*LocalizedString*] | None = None

class clarin.sru.server.config.**IndexInfo**(*sets: List[*clarin.sru.server.config.IndexInfo.Set*]* | *NoneType = None*, *indexes: List[*clarin.sru.server.config.IndexInfo.Index*]* | *NoneType = None*)

Bases: object

class **Set**(*identifier: str*, *name: str*, *title: List[*clarin.sru.server.config.LocalizedString*]* | *NoneType = None*)

Bases: object

identifier: str

name: str

title: List[*LocalizedString*] | None = None

class **Index**(*can_search: bool*, *can_scan: bool*, *can_sort: bool*, *maps: List[*clarin.sru.server.config.IndexInfo.Index.Map*]* | *NoneType = None*, *title: List[*clarin.sru.server.config.LocalizedString*]* | *NoneType = None*)

Bases: object

class **Map**(*primary: bool*, *set: str*, *name: str*)

Bases: object

primary: bool

set: str

name: str

can_search: bool

can_scan: bool

can_sort: bool

maps: List[*Map*] | None = None

title: List[*LocalizedString*] | None = None

sets: List[*Set*] | None = None

indexes: List[*Index*] | None = None

class clarin.sru.server.config.**SRUServerConfig**(*min_version:* SRUVersion, *max_version:* SRUVersion, *default_version:* SRUVersion, *legacy_namespace_mode:* LegacyNamespaceMode, *transport:* *str*, *host:* *str*, *port:* *int*, *database:* *str*, *number_of_records:* *int*, *maximum_records:* *int*, *number_of_terms:* *int*, *maximum_terms:* *int*, *echo_requests:* *bool*, *indent_response:* *int*, *response_buffer_size:* *int*, *allow_override_maximum_records:* *bool*, *allow_override_maximum_terms:* *bool*, *allow_override_indent_response:* *bool*, *database_info:* DatabaseInfo, *index_info:* IndexInfo, *schema_info:* *List*[SchemaInfo] | *None* = *None*)

Bases: object

SRU server configuration.

The XML configuration file must validate against the `sru-server-config.xsd` W3C schema bundled with the package and need to have the `http://www.clarin.eu/sru-server/1.0/` XML namespace.

min_version: *SRUVersion*

max_version: *SRUVersion*

default_version: *SRUVersion*

legacy_namespace_mode: *LegacyNamespaceMode*

transport: str

host: str

port: int

database: str

number_of_records: int

maximum_records: int

number_of_terms: int

maximum_terms: int

echo_requests: bool

indent_response: int

response_buffer_size: int

allow_override_maximum_records: bool

allow_override_maximum_terms: bool

allow_override_indent_response: bool

base_url: str

database_info: *DatabaseInfo*

index_info: *IndexInfo*

schema_info: List[*SchemaInfo*] | None = None

property default_record_xml_escaping: *SRURecordXmlEscaping*

property default_record_packing: *SRURecordPacking*

get_record_schema_identifier(*record_schema_name: str*) → str | None

get_record_schema_name(*schema_identifier: str*) → str | None

find_schema_info(*value: str*) → *SchemaInfo* | None

static find_set_by_name(*sets: List[*Set*]*, *name: str*) → *Set* | None

static fromparams(*params: Dict[str, str]*, *database_info:* DatabaseInfo, *index_info:* IndexInfo | *None* = *None*, *schema_info: List[*SchemaInfo*]* | *None* = *None*) → *SRUServerConfig*

Creates an SRU configuration object with default values and overrides from **params**.

Parameters

- **params** – additional settings
- **database_info** – optinal *DatabaseInfo*
- **index_info** – optinal *IndexInfo*
- **schema_info** – optional list *SchemaInfo*

Returns

SRUServerConfig – a initialized *SRUEndpointConfig* instance

Raises

- **TypeError** – if **params** is None
- ***SRUConfigException*** – if an error occurred

static parse(*params: Dict[str, str]*, *config_file: BytesIO* | *PathLike* | *str*) → *SRUServerConfig*

Parse a SRU server XML configuration file and create an configuration object from it.

Parameters

- **params** – additional settings
- **config_file** – an URL pointing to the XML configuration file

Returns

SRUServerConfig – a initialized *SRUEndpointConfig* instance

Raises

- **TypeError** – if **params** or **configFile** is None
- ***SRUConfigException*** – if an error occurred

static load_config_file(*config_file: BytesIO | PathLike | str*) → _ElementTree

static parse_version(*params: Dict[str, str]*, *name: str*, *mandatory: bool*, *default:* SRUVersion) → *SRUVersion*

static parse_int(*params: Dict[str, str]*, *name: str*, *mandatory: bool*, *default: int*, *min: int*, *max: int*) → int

static parse_bool(*params: Dict[str, str]*, *name: str*, *mandatory: bool*, *default: bool*) → bool

2.7 clarin.sru.server.request

class clarin.sru.server.request.**SRURequest**

Bases: object

Provides information about a SRU request.

abstract get_operation() → *SRUOperation*

Get the operation parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

abstract get_version() → *SRUVersion*

Get the **version** parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

is_version(*version:* SRUVersion) → bool

Check if this request is of a specific version.

Parameters
version – the version to check

Returns

bool –

True if this request is in the requested
version, False otherwise

is_version_between(*min:* SRUVersion, *max:* SRUVersion) → bool

Check if version of this request is at least *min* and at most *max*.

Parameters

- **min** – the minimum version
- **max** – the maximum version

Returns

bool –

True if this request is in the requested
version, False otherwise

abstract get_record_xml_escaping() → *SRURecordXmlEscaping*

Get the **recordXmlEscpaing** (SRU 2.0) or **recordPacking** (SRU 1.1 and SRU 1.2) parameter of this request. Only available for **explain** and **searchRetrieve** requests.

Returns
SRURecordXmlEscaping – the record XML escaping method

abstract get_record_packing() → *SRURecordPacking*

Get the **recordPacking** (SRU 2.0) parameter of this request. Only available for **searchRetrieve** requests.

Returns
SRURecordPacking – the record packing method

abstract get_query() → *SRUQuery*[Any] | None

Get the **query** parameter of this request. Only available for **searchRetrieve** requests.

Returns

SRUQuery[Any] –

an *SRUQuery* instance tailored for the
used queryType or *None* if not a **searchRetrieve** request

get_query_type() → str | None

Get the **queryType** parameter of this request. Only available for **searchRetrieve** requests.

Returns

str –

the queryType of the parsed query or *None* if not a
searchRetrieve request

is_query_type(*query_type: str*) → bool

Check if the request was made with the given queryType. Only available for **searchRetrieve** requests.

Parameters
query_type – the queryType to compare with

Returns

bool –

True if the queryType matches, False
otherwise

abstract get_start_record() → int

Get the **startRecord** parameter of this request. Only available for **searchRetrieve** requests. If the client did not provide a value for the request, it is set to 1.

Returns
int – the number of the start record

abstract get_maximum_records() → int

Get the **maximumRecords** parameter of this request. Only available for **searchRetrieve** requests. If no value was supplied with the request, the server will automatically set a default value.

Returns
int – the maximum number of records

abstract get_record_schema_identifier() → str | None

Get the record schema identifier derived from the **recordSchema** parameter of this request. Only available for **searchRetrieve** requests. If the request was send with the short record schema name, it will automatically expanded to the record schema identifier.

Returns

str –

the record schema identifier or *None* if no
recordSchema parameter was supplied for this request

`abstract get_record_xpath() → str | None`

Get the **recordXPath** parameter of this request. Only available for **searchRetrieve** requests and version 1.1 requests.

Returns

str –

the record XPath or *None* of no value was supplied
for this request

`abstract get_resultSet_TTL() → int`

Get the **resultSetTTL** parameter of this request. Only available for **searchRetrieve** requests.

Returns

int –

the result set TTL or `-1` if no value was
supplied for this request

`abstract get_sortKeys() → str | None`

Get the **sortKeys** parameter of this request. Only available for **searchRetrieve** requests and version 1.1 requests.

Returns

str –

the record XPath or *None* of no value was supplied
for this request

`abstract get_scan_clause() → CQLQuery | None`

Get the **scanClause** parameter of this request. Only available for **scan** requests.

Returns

cql.CQLQuery –

the parsed scan clause or *None* if not a
scan request

`abstract get_response_position() → int`

Get the **responsePosition** parameter of this request. Only available for **scan** requests. If the client did not provide a value for the request, it is set to 1.

Returns
int – the response position

`abstract get_maximum_terms() → int`

Get the **maximumTerms** parameter of this request. Available for any type of request.

Returns

int –

the maximum number of terms or `-1` if no value
was supplied for this request

abstract get_stylesheet() → str | None

Get the **stylesheet** parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Returns

str –

the stylesheet or *None* if no value was supplied
for this request

abstract get_renderBy() → *SRURenderBy* | None

Get the **renderBy** parameter of this request.

Returns

SRURenderBy –

the renderBy parameter or *None* if no value
was supplied for this request

abstract get_response_type() → str | None

(SRU 2.0) The request parameter **responseType**, paired with the Internet media type specified for the response (via either the httpAccept parameter or http accept header) determines the schema for the response.

Returns

str –

the value of the responeType request parameter or
None if no value was supplied for this request

abstract get_http_accept() → str | None

(SRU 2.0) The request parameter **httpAccept** may be supplied to indicate the preferred format of the response. The value is an Internet media type.

Returns

str –

the value of the httpAccept request parameter or
None if no value was supplied for

abstract get_protocol_schema() → str

Get the protocol schema which was used of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Returns
str – the protocol scheme

abstract get_extra_request_data_names() → List[str]

Get the names of extra parameters of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Returns
List[str] – a possibly empty list of parameter names

abstract get_extra_request_data(*name: str*) → str | None

Get the value of an extra parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Parameters
name – name of the extra parameter. Must be prefixed with `x-`

Returns

str –

the value of the parameter of *None* of extra
parameter with that name exists

class clarin.sru.server.request.**ParameterInfo**(*parameter:* clarin.sru.server.request.ParameterInfo.Parameter, *mandatory:* *bool*, *min:* clarin.sru.constants.SRUVersion, *max:* clarin.sru.constants.SRUVersion)

Bases: `object`

class **Parameter**(*value*)

Bases: `str`, `Enum`

An enumeration.

STYLESHEET = 'stylesheet'

RENDER_BY = 'render_by'

HTTP_ACCEPT = 'http_accept'

RESPONSE_TYPE = 'response_type'

START_RECORD = 'start_record'

MAXIMUM_RECORDS = 'maximum_records'

RECORD_XML_ESCAPING = 'record_xml_escaping'

RECORD_PACKING = 'record_packing'

RECORD_SCHEMA = 'record_schema'

RECORD_XPATH = 'record_xpath'

RESULT_SET_TTL = 'result_set_ttl'

SORT_KEYS = 'sort_keys'

SCAN_CLAUSE = 'scan_clause'

RESPONSE_POSITION = 'response_position'

MAXIMUM_TERMS = 'maximum_terms'

parameter: *Parameter*

mandatory: bool

min: *SRUVersion*

max: *SRUVersion*

name(*version:* SRUVersion) → str | None

is_for_version(*version:* SRUVersion) → bool

class clarin.sru.server.request.**ParameterInfoSets**(*value*)

Bases: `Enum`

An enumeration.

EXPLAIN = [ParameterInfo(parameter=<Parameter.STYLESHEET: 'stylesheet'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_1_2: '1.2'>), ParameterInfo(parameter=<Parameter.RECORD_XML_ESCAPING: 'record_xml_escaping'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_1_2: '1.2'>)]

SCAN = [ParameterInfo(parameter=<Parameter.STYLESHEET: 'stylesheet'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.HTTP_ACCEPT: 'http_accept'>, mandatory=False, min=<SRUVersion.VERSION_2_0: '2.0'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.SCAN_CLAUSE: 'scan_clause'>, mandatory=True, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RESPONSE_POSITION: 'response_position'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.MAXIMUM_TERMS: 'maximum_terms'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>)]

SEARCH_RETRIEVE = [ParameterInfo(parameter=<Parameter.STYLESHEET: 'stylesheet'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_1_2: '1.2'>), ParameterInfo(parameter=<Parameter.HTTP_ACCEPT: 'http_accept'>, mandatory=False, min=<SRUVersion.VERSION_2_0: '2.0'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RENDER_BY: 'render_by'>, mandatory=False, min=<SRUVersion.VERSION_2_0: '2.0'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RESPONSE_TYPE: 'response_type'>, mandatory=False, min=<SRUVersion.VERSION_2_0: '2.0'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.START_RECORD: 'start_record'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.MAXIMUM_RECORDS: 'maximum_records'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RECORD_XML_ESCAPING: 'record_xml_escaping'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RECORD_PACKING: 'record_packing'>, mandatory=False, min=<SRUVersion.VERSION_2_0: '2.0'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RECORD_SCHEMA: 'record_schema'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RESULT_SET_TTL: 'result_set_ttl'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>), ParameterInfo(parameter=<Parameter.RECORD_XPATH: 'record_xpath'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_1_2: '1.2'>), ParameterInfo(parameter=<Parameter.SORT_KEYS: 'sort_keys'>, mandatory=False, min=<SRUVersion.VERSION_1_1: '1.1'>, max=<SRUVersion.VERSION_2_0: '2.0'>)]

classmethod for_operation(*operation:* SRUOperation | *None*) → List[*ParameterInfo*] | None

class clarin.sru.server.request.**SRURequestImpl**(*config:* SRUServerConfig, *query_parsers:* SRUQueryParserRegistry, *request:* *Request*, *authentication_info_provider:* SRUAuthenticationInfoProvider | *None* = *None*)

Bases: *SRUDiagnosticList*, *SRURequest*

get_request() → Request

`get_operation()` → *SRUOperation*

Get the `operation` parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

`get_version()` → *SRUVersion*

Get the **version** parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

`get_authentication()` → *SRUAuthenticationInfo* | None

`get_authentication_subject()` → str | None

`get_query()` → *SRUQuery*[Any] | None

Get the **query** parameter of this request. Only available for **searchRetrieve** requests.

Returns

SRUQuery[Any] –

an *SRUQuery* instance tailored for the used queryType or *None* if not a **searchRetrieve** request

`get_record_xml_escaping()` → *SRURecordXmlEscaping*

Get the **recordXmlEscpaing** (SRU 2.0) or **recordPacking** (SRU 1.1 and SRU 1.2) parameter of this request. Only available for **explain** and **searchRetrieve** requests.

Returns

SRURecordXmlEscaping – the record XML escaping method

`get_record_packing()` → *SRURecordPacking*

Get the **recordPacking** (SRU 2.0) parameter of this request. Only available for **searchRetrieve** requests.

Returns

SRURecordPacking – the record packing method

`get_start_record()` → int

Get the **startRecord** parameter of this request. Only available for **searchRetrieve** requests. If the client did not provide a value for the request, it is set to 1.

Returns

int – the number of the start record

`get_maximum_records()` → int

Get the **maximumRecords** parameter of this request. Only available for **searchRetrieve** requests. If no value was supplied with the request, the server will automatically set a default value.

Returns

int – the maximum number of records

`get_record_schema_identifier()` → str | None

Get the record schema identifier derived from the **recordSchema** parameter of this request. Only available for **searchRetrieve** requests. If the request was send with the short record schema name, it will automatically expanded to the record schema identifier.

Returns

str –

the record schema identifier or *None* if no **recordSchema** parameter was supplied for this request

`get_record_xpath()` → str | None

Get the **recordXPath** parameter of this request. Only available for **searchRetrieve** requests and version 1.1 requests.

Returns

str –

the record XPath or *None* of no value was supplied
for this request

`get_resultSet_TTL()` → int

Get the **resultSetTTL** parameter of this request. Only available for **searchRetrieve** requests.

Returns

int –

the result set TTL or `-1` if no value was
supplied for this request

`get_sortKeys()` → str | None

Get the **sortKeys** parameter of this request. Only available for **searchRetrieve** requests and version 1.1 requests.

Returns

str –

the record XPath or *None* of no value was supplied
for this request

`get_scan_clause()` → CQLQuery | None

Get the **scanClause** parameter of this request. Only available for **scan** requests.

Returns

cql.CQLQuery –

the parsed scan clause or *None* if not a
scan request

`get_response_position()` → int

Get the **responsePosition** parameter of this request. Only available for **scan** requests. If the client did not provide a value for the request, it is set to 1.

Returns
int – the response position

`get_maximum_terms()` → int

Get the **maximumTerms** parameter of this request. Available for any type of request.

Returns

int –

the maximum number of terms or `-1` if no value
was supplied for this request

`get_stylesheet()` → str | None

Get the **stylesheet** parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Returns

str –

the stylesheet or *None* if no value was supplied
for this request

`get_renderBy()` → *SRURenderBy* | None

Get the **renderBy** parameter of this request.

Returns

SRURenderBy –

the renderBy parameter or *None* if no value
was supplied for this request

`get_response_type()` → str | None

(SRU 2.0) The request parameter **responseType**, paired with the Internet media type specified for the response (via either the httpAccept parameter or http accept header) determines the schema for the response.

Returns

str –

the value of the responeType request parameter or
None if no value was supplied for this request

`get_version_raw()` → *SRUVersion* | None

`get_record_xml_escaping_raw()` → str | None

`get_record_packing_raw()` → str | None

`get_record_schema_identifier_raw()` → str | None

`get_query_raw()` → str | None

`get_maximum_records_raw()` → int

`get_scan_clause_raw()` → str | None

`get_http_accept_raw()` → str | None

`get_indent_response()` → int

`get_http_accept()` → str | None

(SRU 2.0) The request parameter **httpAccept** may be supplied to indicate the preferred format of the response. The value is an Internet media type.

Returns

str –

the value of the httpAccept request parameter or
None if no value was supplied for

`get_protocol_schema()` → str

Get the protocol schema which was used of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Returns
str – the protocol scheme

`add_diagnostic`(*uri: str*, *details: str | None = None*, *message: str | None = None*) → None

Add a non surrogate diagnostic to the response.

Parameters

- **uri** – the diagnostic's identifying URI
- **details** – supplementary information available, often in a format specified by the diagnostic or `None`
- **message** – human readable message to display to the end user or `None`

`add_diagnostic_obj`(*diagnostic: SRUDiagnostic*)

`check_parameters`() → bool

Validate incoming request parameters

Returns

bool –

`True` **if successful,** `False` **if something** went wrong

`check_parameters_version_operation`() → bool

Validate incoming request parameters **version** and **operation**.

Returns

bool –

`True` **if successful,** `False` **if something** went wrong

`get_parameter_names`() → List[str]

`get_parameter`(*name: SRUParam | str*, *nullify: bool*, *diagnostic_if_empty: bool*) → str | None

`get_extra_request_data_names`() → List[str]

Get the names of extra parameters of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Returns

List[str] – a possibly empty list of parameter names

`get_extra_request_data`(*name: str*) → str | None

Get the value of an extra parameter of this request. Available for **explain**, **searchRetrieve** and **scan** requests.

Parameters

name – name of the extra parameter. Must be prefixed with `x-`

Returns

str –

the value of the parameter of *None* of extra parameter with that name exists

2.8 clarin.sru.server.result

class clarin.sru.server.result.**SRUAbstractResult**(*diagnostics:* SRUDiagnosticList)

Bases: object

Base class for SRU responses.

add_diagnostic(*uri: str*, *details: str | None = None*, *message: str | None = None*) → None

Add a non surrogate diagnostic to the response.

Parameters

- **uri** – the diagnostic's identifying URI
- **details** – supplementary information available, often in a format specified by the diagnostic or None
- **message** – human readable message to display to the end user or None

property has_extra_response_data: bool

Check, if extra response data should be serialized for this request. Default implementation is provided for convince and always returns False.

Returns

bool – True if extra response data should be serialized

write_extra_response_data(*writer:* SRUXMLStreamWriter) → None

Serialize extra response data for this request. A no-op default implementation is provided for convince.

Parameters

writer – Writer to serialize extra response data

close() → None

Release this result and free any associated resources.

This method **must not** throw any exceptions.

Calling the method *close* on a result object that is already closed is a no-op.

class clarin.sru.server.result.**SRUExplainResult**(*diagnostics:* SRUDiagnosticList)

Bases: ABC, *SRUAbstractResult*

A result set of an explain operation. A database implementation may use it implement extensions to the SRU protocol, i.e. providing extraResponseData.

This class needs to be implemented for the target data source.

See also:

SRU Explain Operation: http://www.loc.gov/standards/sru/explain/

class clarin.sru.server.result.**SRUScanResultSet**(*diagnostics:* SRUDiagnosticList)

Bases: ABC, *SRUAbstractResult*

A result set of a scan operation. It is used to iterate over the term set and provides a method to serialize the terms.

A *SRUScanResultSet* object maintains a cursor pointing to its current term. Initially the cursor is positioned before the first term. The *next* method moves the cursor to the next term, and because it returns False when there are no more terms in the *SRUScanResultSet* object, it can be used in a *while* loop to iterate through the term set.

This class needs to be implemented for the target search engine.

See also:

SRU Scan Operation: http://www.loc.gov/standards/sru/companionSpecs/scan.html

class WhereInList(*value*)

Bases: str, Enum

A flag to indicate the position of the term within the complete term list.

FIRST = 'first'

The first term (**first**)

LAST = 'last'

The last term (**last**)

ONLY = 'only'

The only term (**only**)

INNER = 'inner'

Any other term (**inner**)

abstract next_term() → bool

Moves the cursor forward one term from its current position. A result set cursor is initially positioned before the first record; the first call to the method *next* makes the first term the current term; the second call makes the second term the current term, and so on.

When a call to the *next* method returns `False`, the cursor is positioned after the last term.

Returns

bool –

True if the new current term is valid;
`False` if there are no more terms

Raises
SRUException – if an error occurred while fetching the next term

abstract get_value() → str

Get the current term exactly as it appears in the index.

Returns
str – current term

abstract get_number_of_records() → int

Get the number of records for the current term which would be matched if the index in the request's *scanClause* was searched with the term in the *value* field.

Returns

int –

a non-negative number of records or -1, if the
number is unknown.

abstract get_display_term() → str | None

Get the string for the current term to display to the end user in place of the term itself.

Returns
str – display string or `None`

abstract get_WhereInList() → *WhereInList* | None

Get the flag to indicate the position of the term within the complete term list.

Returns
WhereInList – position within term list or `None`

has_extra_term_data() → bool

Check, if extra term data should be serialized for the current term. A default implementation is provided for convince and always returns `False`.

Returns
bool – `True` if the term has extra term data

Raises
`StopIteration` – term set is already advanced past all past terms

See also:

write_extra_term_data

abstract write_extra_term_data(*writer:* SRUXMLStreamWriter)

Serialize extra term data for the current term. A no-op default implementation is provided for convince.

Parameters
writer – Writer to serialize extra term data for current term

Raises
`StopIteration` – term set already advanced past all terms

class clarin.sru.server.result.**SRUSearchResultSet**(*diagnostics:* SRUDiagnosticList)

Bases: `ABC`, *SRUAbstractResult*

A result set of a `searchRetrieve` operation. It it used to iterate over the result set and provides a method to serialize the record in the requested format.

A *SRUSearchResultSet* object maintains a cursor pointing to its current record. Initially the cursor is positioned before the first record. The *next* method moves the cursor to the next record, and because it returns `False` when there are no more records in the *SRUSearchResultSet* object, it can be used in a *while* loop to iterate through the result set.

This class needs to be implemented for the target search engine.

See also:

- SRU Search Retrieve Operation: http://www.loc.gov/standards/sru/
- SRU 1.1 SR: http://www.loc.gov/standards/sru/sru-1-1.html
- SRU 1.2 SR: http://www.loc.gov/standards/sru/sru-1-2.html
- SRU 2.0 SR: http://www.loc.gov/standards/sru/sru-2-0.html
- Differences SRU 2.0 to SRU 1.2: http://www.loc.gov/standards/sru/differences.html

abstract get_total_record_count() → int

The number of records matched by the query. If the query fails this must be `0`. If the search engine cannot determine the total number of matched by a query, it must return `-1`.

Returns

int –

the total number of results or `0` if the query
failed or `-1` if the search engine cannot determine the total number of results

abstract get_record_count() → int

The number of records matched by the query but at most as the number of records requested to be returned (`maximumRecords` parameter). If the query fails this must be `0`.

Returns
int – the number of results or `0` if the query failed

get_resultSet_id() → str | None

The result set id of this result. The default implementation returns `None`.

Returns

str –

the result set id or `None` if not applicable for
this result

get_resultSet_TTL() → int

The result set time to live. In SRU 2.0 it will be serialized as <resultSetTTL> element; in SRU 1.2 as <resultSetIdleTime> element.The default implementation returns `-1`.

Returns

int –

the result set time to live or `-1` if not
applicable for this result

get_result_count_precision() → *SRUResultCountPrecision* | None

(SRU 2.0) Indicate the accuracy of the result count reported by total number of records that matched the query. Default implementation returns `None`.

Returns

Optional[SRUResultCountPrecision] –

the result count
precision or `None` if not applicable for this result

See also:

SRUResultCountPrecision

abstract get_record_schema_identifier() → str

The record schema identifier in which the records are returned (`recordSchema` parameter).

Returns
str – the record schema identifier

abstract next_record() → bool

Moves the cursor forward one record from its current position. A *SRUSearchResultSet* cursor is initially positioned before the first record; the first call to the method *next* makes the first record the current record; the second call makes the second record the current record, and so on.

When a call to the *next* method returns `False`, the cursor is positioned after the last record.

Returns

bool –

True if the new current record is valid;
`False` if there are no more records

Raises
SRUException – if an error occurred while fetching the next record

abstract get_record_identifier() → str | None

An identifier for the current record by which it can unambiguously be retrieved in a subsequent operation.

Returns

str –

identifier for the record or None of none is
available

Raises
StopIteration – result set is past all records

get_surrogate_diagnostic() → *SRUDiagnostic* | None

Get surrogate diagnostic for current record. If this method returns a diagnostic, the *write_record method will not be called. The default implementation returns ``None`.*

Returns

Optional[SRUDiagnostic] –

a surrogate diagnostic or
`None`

abstract write_record(*writer:* SRUXMLStreamWriter) → None

Serialize the current record in the requested format.

Parameters
writer – Writer to serialize current record

Raises
StopIteration – result set is past all records

property has_extra_record_data: bool

Check, if extra record data should be serialized for the current record. The default implementation returns `False`.

Returns
bool – `True` if the record has extra record data

Raises
StopIteration – result set is past all records

See also:

write_extra_record_data

write_extra_record_data(*writer:* SRUXMLStreamWriter) → None

Serialize extra record data for the current record. A no-op default implementation is provided for convince.

Parameters
writer – Writer to serialize extra record data for current record

Raises
StopIteration – result set past already advanced past all records

2.9 clarin.sru.server.server

class clarin.sru.server.server.**SRUNamespaces**(*response_NS: str*, *response_prefix: str*, *scan_NS: str*, *scan_prefix: str*, *diagnostic_NS: str*, *XCQL_NS: str*, *diagnostic_prefix: str = 'diag'*, *explain_NS: str = 'http://explain.z3950.org/dtd/2.0/'*, *explain_prefix: str = 'zr'*)

Bases: object

Interface for decoupling SRU namespaces from implementation to allow to support SRU 1.1/1.2 and SRU 2.0.

response_NS: str

The namespace URI for encoding **explain** and **searchRetrieve** operation responses.

response_prefix: str

The namespace prefix for encoding **explain** and **searchRetrieve**

scan_NS: str

The namespace URI for encoding **scan** operation responses.

scan_prefix: str

The namespace prefix for encoding **scan** operation responses.

diagnostic_NS: str

The namespace URI for encoding SRU diagnostics.

XCQL_NS: str

The namespace URI for encoding XCQL fragments

diagnostic_prefix: str = 'diag'

The namespace prefix for encoding SRU diagnostics.

explain_NS: str = 'http://explain.z3950.org/dtd/2.0/'

The namespace URI for encoding explain record data fragments.

explain_prefix: str = 'zr'

The namespace prefix for encoding explain record data fragments.

static for_legacy_LOC() → *SRUNamespaces*

static for_1_2_OASIS() → *SRUNamespaces*

static for_2_0() → *SRUNamespaces*

static get_namespaces(*version:* SRUVersion, *legacy_ns_mode:* LegacyNamespaceMode) → *SRUNamespaces*

class clarin.sru.server.server.**SRUSearchEngine**

Bases: object

Interface for connecting the SRU protocol implementation to an actual search engine. Base class required for an *SRUSearchEngine* implementation to be used with the *SRUServerApp*.

Implementing the *explain* and *scan* is optional, but implementing *search* is mandatory.

The implementation of these methods **must** be thread-safe.

abstract explain(*config:* SRUServerConfig, *request:* SRURequest, *diagnostics:* SRUDiagnosticList) → *SRUExplainResult* | None

Handle an **explain** operation. Implementing this method is optional, but is required, if the **writeExtraResponseData** block of the SRU response needs to be filled. The arguments for this operation are provides by the *SRURequest* object.

The implementation of this method **must** be thread-safe.

Parameters

- **config** – the *SRUEndpointConfig* object that contains the endpoint configuration
- **request** – the *SRURequest* object that contains the request made to the endpoint
- **diagnostics** – the *SRUDiagnosticList* object for storing non-fatal diagnostics

Returns

SRUExplainResult –

a *SRUExplainResult* object or `None`
if the search engine does not want to provide *write_extra_response_data*

Raises
SRUException – if an fatal error occurred

abstract search(*config:* SRUServerConfig, *request:* SRURequest, *diagnostics:* SRUDiagnosticList) → *SRUSearchResultSet*

Handle a **searchRetrieve** operation. Implementing this method is mandatory. The arguments for this operation are provides by the *SRURequest* object.

The implementation of this method **must** be thread-safe.

Parameters

- **config** – the *SRUEndpointConfig* object that contains the endpoint configuration
- **request** – the *SRURequest* object that contains the request made to the endpoint
- **diagnostics** – the *SRUDiagnosticList* object for storing non-fatal diagnostics

Returns
SRUSearchResultSet – a *SRUSearchResultSet* object

Raises
SRUException – if an fatal error occurred

abstract scan(*config:* SRUServerConfig, *request:* SRURequest, *diagnostics:* SRUDiagnosticList) → *SRUScanResultSet* | None

Handle a **scan** operation. Implementing this method is optional. If you don't need to handle the **scan** operation, just return `None` and the SRU server will return the appropiate diagnostic to the client. The arguments for this operation are provides by the *SRURequest* object.

The implementation of this method **must** be thread-safe.

Parameters

- **config** – the *SRUEndpointConfig* object that contains the endpoint configuration
- **request** – the *SRURequest* object that contains the request made to the endpoint
- **diagnostics** – the *SRUDiagnosticList* object for storing non-fatal diagnostics

Returns

SRUScanResultSet –

a *SRUScanResultSet* object or `None`
if this operation is not supported by this search engine

Raises
SRUException – if an fatal error occurred

init(*config:* SRUServerConfig, *query_parser_registry_builder:* Builder, *params:* *Dict[str, str]*) → None

Initialize the search engine.

Parameters

- **config** – the *SRUServerConfig* object for this search engine
- **query_parser_registry_builder** – the *SRUQueryParserRegistry.Builder* object to be used for this search engine. Use to register additional query parsers with the *SRUServer*
- **params** – additional parameters from the server

Raises
SRUConfigException – an error occurred during initialization of the search engine

destroy() → None

Destroy the search engine. Use this method for any cleanup the search engine needs to perform upon termination.

class clarin.sru.server.server.**SRUServer**(*config:* SRUServerConfig, *query_parsers:* SRUQueryParserRegistry, *search_engine:* SRUSearchEngine, *authentication_info_provider:* SRUAuthenticationInfoProvider | *None* = *None*)

Bases: object

SRU/CQL protocol implementation for the server-side (SRU/S). This class implements SRU/CQL version 1.1 and and 1.2.

See also:

SRU/CQL protocol 1.2: http://www.loc.gov/standards/sru/

handle_request(*request: Request*, *response: Response*)

Handle a SRU request.

TEMP_OUTPUT_BUFFERING = False

explain(*request:* SRURequestImpl, *response: Response*)

scan(*request:* SRURequestImpl, *response: Response*)

search(*request:* SRURequestImpl, *response: Response*)

2.10 clarin.sru.server.wsgi

class clarin.sru.server.wsgi.**SRUServerApp**(*SRUSearchEngine_clazz: Type[*SRUSearchEngine*]* | SRUSearchEngine, *config_file: str*, *params:* *Dict[*SRUServerConfigKey | *str, str]*, *develop: bool* = *False*)

Bases: object

set_default_params() → None

init() → None

destroy() → None

Destroy the SRU server application

wsgi_app(*environ: WSGIEnvironment*, *start_response: StartResponse*) → Iterable[bytes]

2.11 clarin.sru.xml.writer

class clarin.sru.xml.writer.**SRUXMLStreamWriter**(*output_stream: TextIOBase*, *record_escaping:* SRURecordXmlEscaping, *indent: int = -1*, *encoding: str = 'utf-8'*, *short_empty_elements: bool = False*)

Bases: ContentHandler

class IndentingState(*value*)

Bases: Enum

An enumeration.

SEEN_NOTHING = 1

SEEN_ELEMENT = 2

SEEN_DATA = 3

onStartElement()

onEndElement()

onEmptyElement()

doIndent()

startRecord()

endRecord()

setDocumentLocator(*locator*)

Called by the parser to give the application a locator for locating the origin of document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

startPrefixMapping(*prefix*, *uri*)

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the http://xml.org/sax/features/namespaces feature is true (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding startElement event, and all endPrefixMapping events will occur after the corresponding endElement event, but their order is not guaranteed.

endPrefixMapping(*prefix*)

End the scope of a prefix-URI mapping.

See startPrefixMapping for details. This event will always occur after the corresponding endElement event, but the order of endPrefixMapping events is not otherwise guaranteed.

processingInstruction(*target*, *data*)

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

startDocument()

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTDHandler (except for setDocumentLocator).

endDocument()

Receive notification of the end of a document.

The SAX parser will invoke this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

startElement(*name*, *attrs=None*)

Signals the start of an element in non-namespace mode.

The name parameter contains the raw XML 1.0 name of the element type as a string and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

endElement(*name*)

Signals the end of an element in non-namespace mode.

The name parameter contains the name of the element type, just as with the startElement event.

startElementNS(*name*, *qname=None*, *attrs=None*)

Signals the start of an element in namespace mode.

The name parameter contains the name of the element type as a (uri, localname) tuple, the qname parameter the raw XML 1.0 name used in the source document, and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

The uri part of the name tuple is None for elements which have no namespace.

endElementNS(*name*, *qname=None*)

Signals the end of an element in namespace mode.

The name parameter contains the name of the element type, just as with the startElementNS event.

characters(*content*)

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

ignorableWhitespace(*whitespace*)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

skippedEntity(*name*)

Receive notification of a skipped entity.

The Parser will invoke this method once for each entity skipped. Non-validating processors may skip entities if they have not seen the declarations (because, for example, the entity was declared in an external DTD subset). All processors may skip external entities, depending on the values of the http://xml.org/sax/features/external-general-entities and the http://xml.org/sax/features/external-parameter-entities properties.

writeXCQL(*query: CQLQuery*, *search_retrieve_mode: bool*)

prefix(*prefix*, *uri*)

element(*name*, *namespace=None*, *attrs=None*)

elementcontent(*name*, *content=None*, *namespace=None*, *attrs=None*)

record()

clarin.sru.xml.writer.**copy_XML_into_writer**(*writer: ContentHandler*, *xml: bytes | str*)

class clarin.sru.xml.writer.**XMLStreamWriterHelper**(*xmlwriter: ContentHandler*)

Bases: ContentHandler

setDocumentLocator(*locator*)

Called by the parser to give the application a locator for locating the origin of document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

startPrefixMapping(*prefix*, *uri*)

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the http://xml.org/sax/features/namespaces feature is true (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding startElement event, and all endPrefixMapping events will occur after the corresponding endElement event, but their order is not guaranteed.

endPrefixMapping(*prefix*)

End the scope of a prefix-URI mapping.

See startPrefixMapping for details. This event will always occur after the corresponding endElement event, but the order of endPrefixMapping events is not otherwise guaranteed.

processingInstruction(*target*, *data*)

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

startDocument()

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTDHandler (except for setDocumentLocator).

endDocument()

Receive notification of the end of a document.

The SAX parser will invoke this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

startElement(*name*, *attrs=None*)

Signals the start of an element in non-namespace mode.

The name parameter contains the raw XML 1.0 name of the element type as a string and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

endElement(*name*)

Signals the end of an element in non-namespace mode.

The name parameter contains the name of the element type, just as with the startElement event.

startElementNS(*name*, *qname=None*, *attrs=None*)

Signals the start of an element in namespace mode.

The name parameter contains the name of the element type as a (uri, localname) tuple, the qname parameter the raw XML 1.0 name used in the source document, and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

The uri part of the name tuple is None for elements which have no namespace.

endElementNS(*name*, *qname=None*)

Signals the end of an element in namespace mode.

The name parameter contains the name of the element type, just as with the startElementNS event.

characters(*content*)

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

ignorableWhitespace(*whitespace*)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

skippedEntity(*name*)

Receive notification of a skipped entity.

The Parser will invoke this method once for each entity skipped. Non-validating processors may skip entities if they have not seen the declarations (because, for example, the entity was declared in an external DTD subset). All processors may skip external entities, depending on the values of the http://xml.org/sax/features/external-general-entities and the http://xml.org/sax/features/external-parameter-entities properties.

writeXML(*xml: bytes | str*)

writeXMLdocument(*xmldoc: Element*)

prefix(*prefix*, *uri*)

element(*name*, *namespace=None*, *attrs=None*)

elementcontent(*name*, *content=None*, *namespace=None*, *attrs=None*)

startRecord()

endRecord()

record()

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

C

clarin.sru.constants, 5
clarin.sru.diagnostic, 11
clarin.sru.exception, 12
clarin.sru.queryparser, 12
clarin.sru.server.auth, 15
clarin.sru.server.config, 16
clarin.sru.server.request, 22
clarin.sru.server.result, 32
clarin.sru.server.server, 37
clarin.sru.server.wsgi, 39
clarin.sru.xml.writer, 40

INDEX

A

add_diagnostic() (*clarin.sru.diagnostic.SRUDiagnosticList method*), 11
add_diagnostic() (*clarin.sru.server.request.SRURequestImpl method*), 30
add_diagnostic() (*clarin.sru.server.result.SRUAbstractResult method*), 32
add_diagnostic_obj() (*clarin.sru.server.request.SRURequestImpl method*), 31
allow_override_indent_response (*clarin.sru.server.config.SRUServerConfig attribute*), 21
allow_override_maximum_records (*clarin.sru.server.config.SRUServerConfig attribute*), 21
allow_override_maximum_terms (*clarin.sru.server.config.SRUServerConfig attribute*), 21
ANCHORING_CHARACTER_IN_UNSUPPORTED_POSITION (*clarin.sru.constants.SRUDiagnostics attribute*), 8
ANCHORING_CHARACTER_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
AUTHENTICATION_ERROR (*clarin.sru.constants.SRUDiagnostics attribute*), 7
authentication_method (*clarin.sru.server.auth.SRUAuthenticationInfo property*), 15
author (*clarin.sru.server.config.DatabaseInfo attribute*), 18

B

base_url (*clarin.sru.server.config.SRUServerConfig attribute*), 21
build() (*clarin.sru.queryparser.SRUQueryParserRegistry.Builder method*), 14

C

can_scan (*clarin.sru.server.config.IndexInfo.Index attribute*), 19
can_search (*clarin.sru.server.config.IndexInfo.Index attribute*), 19
can_sort (*clarin.sru.server.config.IndexInfo.Index attribute*), 20
CANNOT_PROCESS_QUERY_REASON_UNKNOWN (*clarin.sru.constants.SRUDiagnostics attribute*), 8
CANNOT_SORT_INCOMPATIBLE_RECORD_FORMATS (*clarin.sru.constants.SRUDiagnostics attribute*), 9
category (*clarin.sru.constants.SRUDiagnostics attribute*), 7
characters() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
characters() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
check_parameters() (*clarin.sru.server.request.SRURequestImpl method*), 31
check_parameters_version_operation() (*clarin.sru.server.request.SRURequestImpl method*), 31
clarin.sru.constants
    module, 5
clarin.sru.diagnostic
    module, 11
clarin.sru.exception
    module, 12
clarin.sru.queryparser
    module, 12
clarin.sru.server.auth
    module, 15
clarin.sru.server.config
    module, 16
clarin.sru.server.request
    module, 22
clarin.sru.server.result
    module, 32
clarin.sru.server.server
    module, 37
clarin.sru.server.wsgi
    module, 39

clarin.sru.xml.writer
module, 40
CLIENT (*clarin.sru.constants.SRURenderBy attribute*), 6
close() (*clarin.sru.server.result.SRUAbstractResult method*), 32
COMBINATION_OF_PROXIMITY_ADJACENCY_AND_ANCHORING_CHARACTERS_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
COMBINATION_OF_PROXIMITY_ADJACENCY_AND_MASKING_CHARACTERS_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
COMBINATION_OF_RESULT_SETS_WITH_SEARCH_TERMS_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
copy_XML_into_writer() (*in module clarin.sru.xml.writer*), 42
CQL (*clarin.sru.constants.SRUQueryType attribute*), 5
CQLQuery (*class in clarin.sru.queryparser*), 14
CQLQueryParser (*class in clarin.sru.queryparser*), 14
create_SRUAuthenticationInfoProvider() (*clarin.sru.server.auth.SRUAuthenticationInfoProviderFactory method*), 15
CURRENT (*clarin.sru.constants.SRUResultCountPrecision attribute*), 6

D

database (*clarin.sru.server.config.SRUServerConfig attribute*), 20
DATABASE_DOES_NOT_EXIST (*clarin.sru.constants.SRUDiagnostics attribute*), 7
database_info (*clarin.sru.server.config.SRUServerConfig attribute*), 21
DatabaseInfo (*class in clarin.sru.server.config*), 18
default_record_packing (*clarin.sru.server.config.SRUServerConfig property*), 21
default_record_xml_escaping (*clarin.sru.server.config.SRUServerConfig property*), 21
default_version (*clarin.sru.server.config.SRUServerConfig attribute*), 20
description (*clarin.sru.constants.SRUDiagnostics attribute*), 7
description (*clarin.sru.server.config.DatabaseInfo attribute*), 18
destroy() (*clarin.sru.server.server.SRUSearchEngine method*), 39
destroy() (*clarin.sru.server.wsgi.SRUServerApp method*), 40
details (*clarin.sru.diagnostic.SRUDiagnostic attribute*), 11
diagnostic_NS (*clarin.sru.server.server.SRUNamespaces attribute*), 37
diagnostic_prefix (*clarin.sru.server.server.SRUNamespaces attribute*), 37
doIndent() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40

E

echo_requests (*clarin.sru.server.config.SRUServerConfig attribute*), 20
element() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 42
element() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
elementcontent() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 42
elementcontent() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
EMPTY_TERM_UNSUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
endDocument() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
endDocument() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
endElement() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
endElement() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
endElementNS() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
endElementNS() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
endPrefixMapping() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
endPrefixMapping() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
endRecord() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40
endRecord() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
ESTIMATE (*clarin.sru.constants.SRUResultCountPrecision attribute*), 6
EXACT (*clarin.sru.constants.SRUResultCountPrecision attribute*), 6
EXPLAIN (*clarin.sru.constants.SRUOperation attribute*), 5
EXPLAIN (*clarin.sru.server.request.ParameterInfoSets attribute*), 26
explain() (*clarin.sru.server.server.SRUSearchEngine method*), 37
explain() (*clarin.sru.server.server.SRUServer method*), 39
explain_NS (*clarin.sru.server.server.SRUNamespaces attribute*), 37

explain_prefix (*clarin.sru.server.server.SRUNamespaces attribute*), 37
extent (*clarin.sru.server.config.DatabaseInfo attribute*), 18

F

find_query_parser() (*clarin.sru.queryparser.SRUQueryParserRegistry method*), 13
find_schema_info() (*clarin.sru.server.config.SRUServerConfig method*), 21
find_set_by_name() (*clarin.sru.server.config.SRUServerConfig static method*), 21
FIRST (*clarin.sru.server.result.SRUScanResultSet.WhereInList attribute*), 33
FIRST_RECORD_POSITION_OUT_OF_RANGE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
for_1_2_OASIS() (*clarin.sru.server.server.SRUNamespaces static method*), 37
for_2_0() (*clarin.sru.server.server.SRUNamespaces static method*), 37
for_legacy_LOC() (*clarin.sru.server.server.SRUNamespaces static method*), 37
for_operation() (*clarin.sru.server.request.ParameterInfoSets class method*), 27
fromparams() (*clarin.sru.server.config.SRUServerConfig static method*), 21

G

GENERAL_SYSTEM_ERROR (*clarin.sru.constants.SRUDiagnostics attribute*), 7
get_authentication() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_authentication_subject() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_AuthenticationInfo() (*clarin.sru.server.auth.SRUAuthenticationInfoProvider method*), 15
get_by_uri() (*clarin.sru.constants.SRUDiagnostics class method*), 10
get_default_error_message() (*clarin.sru.diagnostic.SRUDiagnostic static method*), 11
get_diagnostic() (*clarin.sru.exception.SRUException method*), 12
get_display_term() (*clarin.sru.server.result.SRUScanResultSet method*), 33
get_extra_request_data() (*clarin.sru.server.request.SRURequest method*), 25
get_extra_request_data() (*clarin.sru.server.request.SRURequestImpl method*), 31
get_extra_request_data_names() (*clarin.sru.server.request.SRURequest method*), 25
get_extra_request_data_names() (*clarin.sru.server.request.SRURequestImpl method*), 31
get_http_accept() (*clarin.sru.server.request.SRURequest method*), 25
get_http_accept() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_http_accept_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_indent_response() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_maximum_records() (*clarin.sru.server.request.SRURequest method*), 23
get_maximum_records() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_maximum_records_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_maximum_terms() (*clarin.sru.server.request.SRURequest method*), 24
get_maximum_terms() (*clarin.sru.server.request.SRURequestImpl method*), 29
get_namespaces() (*clarin.sru.server.server.SRUNamespaces static method*), 37
get_number_of_records() (*clarin.sru.server.result.SRUScanResultSet method*), 33
get_operation() (*clarin.sru.server.request.SRURequest method*), 22
get_operation() (*clarin.sru.server.request.SRURequestImpl method*), 27
get_parameter() (*clarin.sru.server.request.SRURequestImpl method*), 31
get_parameter_names() (*clarin.sru.server.request.SRURequestImpl method*), 31
get_protocol_schema() (*clarin.sru.server.request.SRURequest method*), 25
get_protocol_schema() (*clarin.sru.server.request.SRURequestImpl method*), 30

get_query() (*clarin.sru.server.request.SRURequest method*), 23
get_query() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_query_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_query_type() (*clarin.sru.server.request.SRURequest method*), 23
get_record_count() (*clarin.sru.server.result.SRUSearchResultSet method*), 35
get_record_identifier() (*clarin.sru.server.result.SRUSearchResultSet method*), 36
get_record_packing() (*clarin.sru.server.request.SRURequest method*), 23
get_record_packing() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_record_packing_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_record_schema_identifier() (*clarin.sru.server.config.SRUServerConfig method*), 21
get_record_schema_identifier() (*clarin.sru.server.request.SRURequest method*), 23
get_record_schema_identifier() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_record_schema_identifier() (*clarin.sru.server.result.SRUSearchResultSet method*), 35
get_record_schema_identifier_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_record_schema_name() (*clarin.sru.server.config.SRUServerConfig method*), 21
get_record_xml_escaping() (*clarin.sru.server.request.SRURequest method*), 22
get_record_xml_escaping() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_record_xml_escaping_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_record_xpath() (*clarin.sru.server.request.SRURequest method*), 24
get_record_xpath() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_renderBy() (*clarin.sru.server.request.SRURequest method*), 25
get_renderBy() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_request() (*clarin.sru.server.request.SRURequestImpl method*), 27
get_response_position() (*clarin.sru.server.request.SRURequest method*), 24
get_response_position() (*clarin.sru.server.request.SRURequestImpl method*), 29
get_response_type() (*clarin.sru.server.request.SRURequest method*), 25
get_response_type() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_result_count_precision() (*clarin.sru.server.result.SRUSearchResultSet method*), 35
get_resultSet_id() (*clarin.sru.server.result.SRUSearchResultSet method*), 35
get_resultSet_TTL() (*clarin.sru.server.request.SRURequest method*), 24
get_resultSet_TTL() (*clarin.sru.server.request.SRURequestImpl method*), 29
get_resultSet_TTL() (*clarin.sru.server.result.SRUSearchResultSet method*), 35
get_scan_clause() (*clarin.sru.server.request.SRURequest method*), 24
get_scan_clause() (*clarin.sru.server.request.SRURequestImpl method*), 29
get_scan_clause_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_sortKeys() (*clarin.sru.server.request.SRURequest method*), 24
get_sortKeys() (*clarin.sru.server.request.SRURequestImpl method*), 29
get_start_record() (*clarin.sru.server.request.SRURequest method*), 23
get_start_record() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_stylesheet() (*clarin.sru.server.request.SRURequest method*), 24
get_stylesheet() (*clarin.sru.server.request.SRURequestImpl method*), 29
get_surrogate_diagnostic() (*clarin.sru.server.result.SRUSearchResultSet method*), 36
get_total_record_count()

(*clarin.sru.server.result.SRUSearchResultSet method*), 34
get_value() (*clarin.sru.server.result.SRUScanResultSet method*), 33
get_version() (*clarin.sru.server.request.SRURequest method*), 22
get_version() (*clarin.sru.server.request.SRURequestImpl method*), 28
get_version_raw() (*clarin.sru.server.request.SRURequestImpl method*), 30
get_WhereInList() (*clarin.sru.server.result.SRUScanResultSet method*), 33

H

handle_request() (*clarin.sru.server.server.SRUServer method*), 39
has_extra_record_data (*clarin.sru.server.result.SRUSearchResultSet property*), 36
has_extra_response_data (*clarin.sru.server.result.SRUAbstractResult property*), 32
has_extra_term_data() (*clarin.sru.server.result.SRUScanResultSet method*), 34
history (*clarin.sru.server.config.DatabaseInfo attribute*), 18
host (*clarin.sru.server.config.SRUServerConfig attribute*), 20
HTTP_ACCEPT (*clarin.sru.constants.SRUParam attribute*), 10
HTTP_ACCEPT (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26

I

identifier (*clarin.sru.server.config.IndexInfo.Set attribute*), 19
identifier (*clarin.sru.server.config.SchemaInfo attribute*), 19
ignorableWhitespace() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 42
ignorableWhitespace() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
implementation (*clarin.sru.server.config.DatabaseInfo attribute*), 19
indent_response (*clarin.sru.server.config.SRUServerConfig attribute*), 20
index_info (*clarin.sru.server.config.SRUServerConfig attribute*), 21
indexes (*clarin.sru.server.config.IndexInfo attribute*), 20
IndexInfo (*class in clarin.sru.server.config*), 19
IndexInfo.Index (*class in clarin.sru.server.config*), 19
IndexInfo.Index.Map (*class in clarin.sru.server.config*), 19
IndexInfo.Set (*class in clarin.sru.server.config*), 19
init() (*clarin.sru.server.server.SRUSearchEngine method*), 39
init() (*clarin.sru.server.wsgi.SRUServerApp method*), 39
INNER (*clarin.sru.server.result.SRUScanResultSet.WhereInList attribute*), 33
INVALID_OR_UNSUPPORTED_USE_OF_PARENTHESES (*clarin.sru.constants.SRUDiagnostics attribute*), 7
INVALID_OR_UNSUPPORTED_USE_OF_QUOTES (*clarin.sru.constants.SRUDiagnostics attribute*), 7
is_for_version() (*clarin.sru.server.request.ParameterInfo method*), 26
is_query_type() (*clarin.sru.server.request.SRURequest method*), 23
is_version() (*clarin.sru.server.request.SRURequest method*), 22
is_version_between() (*clarin.sru.server.request.SRURequest method*), 22

L

lang (*clarin.sru.server.config.LocalizedString attribute*), 16
langUsage (*clarin.sru.server.config.DatabaseInfo attribute*), 19
LAST (*clarin.sru.server.result.SRUScanResultSet.WhereInList attribute*), 33
legacy_namespace_mode (*clarin.sru.server.config.SRUServerConfig attribute*), 20
LegacyNamespaceMode (*class in clarin.sru.server.config*), 16
links (*clarin.sru.server.config.DatabaseInfo attribute*), 19
load_config_file() (*clarin.sru.server.config.SRUServerConfig static method*), 22
LOC (*clarin.sru.server.config.LegacyNamespaceMode attribute*), 16
LocalizedString (*class in clarin.sru.server.config*), 16
location (*clarin.sru.server.config.SchemaInfo attribute*), 19

M

major (*clarin.sru.constants.SRUVersion attribute*), 6
mandatory (*clarin.sru.server.request.ParameterInfo attribute*), 26
MANDATORY_PARAMETER_NOT_SUPPLIED (*clarin.sru.constants.SRUDiagnostics attribute*), 7

maps (*clarin.sru.server.config.IndexInfo.Index attribute*), 20
MASKED_WORDS_TOO_SHORT (*clarin.sru.constants.SRUDiagnostics attribute*), 8
MASKING_CHARACTER_IN_UNSUPPORTED_POSITION (*clarin.sru.constants.SRUDiagnostics attribute*), 8
MASKING_CHARACTER_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
max (*clarin.sru.server.request.ParameterInfo attribute*), 26
max_version (*clarin.sru.server.config.SRUServerConfig attribute*), 20
MAXIMUM (*clarin.sru.constants.SRUResultCountPrecision attribute*), 6
MAXIMUM_RECORDS (*clarin.sru.constants.SRUParam attribute*), 10
maximum_records (*clarin.sru.server.config.SRUServerConfig attribute*), 20
MAXIMUM_RECORDS (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
MAXIMUM_TERMS (*clarin.sru.constants.SRUParam attribute*), 10
maximum_terms (*clarin.sru.server.config.SRUServerConfig attribute*), 20
MAXIMUM_TERMS (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
message (*clarin.sru.diagnostic.SRUDiagnostic attribute*), 11
min (*clarin.sru.server.request.ParameterInfo attribute*), 26
min_version (*clarin.sru.server.config.SRUServerConfig attribute*), 20
MINIMUM (*clarin.sru.constants.SRUResultCountPrecision attribute*), 6
minor (*clarin.sru.constants.SRUVersion attribute*), 6
module
    clarin.sru.constants, 5
    clarin.sru.diagnostic, 11
    clarin.sru.exception, 12
    clarin.sru.queryparser, 12
    clarin.sru.server.auth, 15
    clarin.sru.server.config, 16
    clarin.sru.server.request, 22
    clarin.sru.server.result, 32
    clarin.sru.server.server, 37
    clarin.sru.server.wsgi, 39
    clarin.sru.xml.writer, 40

N

name (*clarin.sru.server.config.IndexInfo.Index.Map attribute*), 19
name (*clarin.sru.server.config.IndexInfo.Set attribute*), 19
name (*clarin.sru.server.config.SchemaInfo attribute*), 19
name() (*clarin.sru.server.request.ParameterInfo method*), 26
next_record() (*clarin.sru.server.result.SRUSearchResultSet method*), 35
next_term() (*clarin.sru.server.result.SRUScanResultSet method*), 33
NON_SPECIAL_CHARACTER_ESCAPED_IN_TERM (*clarin.sru.constants.SRUDiagnostics attribute*), 7
NOT_AUTHORISED_TO_SEND_RECORD (*clarin.sru.constants.SRUDiagnostics attribute*), 9
NOT_AUTHORISED_TO_SEND_RECORD_IN_THIS_SCHEMA (*clarin.sru.constants.SRUDiagnostics attribute*), 9
nr (*clarin.sru.constants.SRUDiagnostics attribute*), 7
number_of_records (*clarin.sru.server.config.SRUServerConfig attribute*), 20
number_of_terms (*clarin.sru.server.config.SRUServerConfig attribute*), 20

O

OASIS (*clarin.sru.server.config.LegacyNamespaceMode attribute*), 16
onEmptyElement() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40
onEndElement() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40
ONLY (*clarin.sru.server.result.SRUScanResultSet.WhereInList attribute*), 33
onStartElement() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40
OP_EXPLAIN (*clarin.sru.constants.SRUParamValue attribute*), 11
OP_SCAN (*clarin.sru.constants.SRUParamValue attribute*), 11
OP_SEARCH_RETRIEVE (*clarin.sru.constants.SRUParamValue attribute*), 11
OPERATION (*clarin.sru.constants.SRUParam attribute*), 10

P

PACKED (*clarin.sru.constants.SRURecordPacking attribute*), 5
parameter (*clarin.sru.server.request.ParameterInfo attribute*), 26
ParameterInfo (*class in clarin.sru.server.request*), 26
ParameterInfo.Parameter (*class in clarin.sru.server.request*), 26
ParameterInfoSets (*class in clarin.sru.server.request*), 26

parse() (*clarin.sru.server.config.SRUServerConfig static method*), 21
parse_bool() (*clarin.sru.server.config.SRUServerConfig static method*), 22
parse_int() (*clarin.sru.server.config.SRUServerConfig static method*), 22
parse_query() (*clarin.sru.queryparser.CQLQueryParser method*), 15
parse_query() (*clarin.sru.queryparser.SearchTermsQueryParser method*), 14
parse_query() (*clarin.sru.queryparser.SRUQueryParser method*), 13
parse_version() (*clarin.sru.server.config.SRUServerConfig static method*), 22
parsed_query (*clarin.sru.queryparser.SRUQuery property*), 12
PATH_UNSUPPORTED_FOR_SCHEMA (*clarin.sru.constants.SRUDiagnostics attribute*), 9
port (*clarin.sru.server.config.SRUServerConfig attribute*), 20
prefix() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 42
prefix() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
primary (*clarin.sru.server.config.IndexInfo.Index.Map attribute*), 19
primary (*clarin.sru.server.config.LocalizedString attribute*), 16
processingInstruction() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
processingInstruction() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
PROXIMITY_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8

Q

QUERY (*clarin.sru.constants.SRUParam attribute*), 10
QUERY_FEATURE_UNSUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
query_parameter_names (*clarin.sru.queryparser.CQLQueryParser property*), 15
query_parameter_names (*clarin.sru.queryparser.SearchTermsQueryParser property*), 14
query_parameter_names (*clarin.sru.queryparser.SRUQueryParser property*), 13
query_parsers (*clarin.sru.queryparser.SRUQueryParserRegistry property*), 13
QUERY_SYNTAX_ERROR (*clarin.sru.constants.SRUDiagnostics attribute*), 7
QUERY_TYPE (*clarin.sru.constants.SRUParam attribute*), 10
query_type (*clarin.sru.queryparser.CQLQuery property*), 14
query_type (*clarin.sru.queryparser.CQLQueryParser property*), 15
query_type (*clarin.sru.queryparser.SearchTermsQuery property*), 14
query_type (*clarin.sru.queryparser.SearchTermsQueryParser property*), 14
query_type (*clarin.sru.queryparser.SRUQuery property*), 12
query_type (*clarin.sru.queryparser.SRUQueryParser property*), 12
query_type_definition (*clarin.sru.queryparser.SRUQueryParser property*), 13

R

raw_query (*clarin.sru.queryparser.SRUQuery property*), 12
record() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 42
record() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
RECORD_DOES_NOT_EXIST (*clarin.sru.constants.SRUDiagnostics attribute*), 9
RECORD_NOT_AVAILABLE_IN_THIS_SCHEMA (*clarin.sru.constants.SRUDiagnostics attribute*), 9
RECORD_PACKING (*clarin.sru.constants.SRUParam attribute*), 10
RECORD_PACKING (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
RECORD_PACKING_PACKED (*clarin.sru.constants.SRUParamValue attribute*), 11
RECORD_PACKING_UNPACKED (*clarin.sru.constants.SRUParamValue attribute*), 11
RECORD_SCHEMA (*clarin.sru.constants.SRUParam attribute*), 10
RECORD_SCHEMA (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
RECORD_TEMPORARILY_UNAVAILABLE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
RECORD_TOO_LARGE_TO_SEND (*clarin.sru.constants.SRUDiagnostics at-*

tribute), 9
RECORD_XML_ESCAPING (*clarin.sru.constants.SRUParam attribute*), 10
RECORD_XML_ESCAPING (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
RECORD_XML_ESCAPING_STRING (*clarin.sru.constants.SRUParamValue attribute*), 11
RECORD_XML_ESCAPING_XML (*clarin.sru.constants.SRUParamValue attribute*), 11
RECORD_XPATH (*clarin.sru.constants.SRUParam attribute*), 10
RECORD_XPATH (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
register() (*clarin.sru.queryparser.SRUQueryParserRegistry.Builder method*), 14
register_defaults() (*clarin.sru.queryparser.SRUQueryParserRegistry.Builder method*), 14
RENDER_BY (*clarin.sru.constants.SRUParam attribute*), 10
RENDER_BY (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
RENDER_BY_CLIENT (*clarin.sru.constants.SRUParamValue attribute*), 11
RENDER_BY_SERVER (*clarin.sru.constants.SRUParamValue attribute*), 11
response_buffer_size (*clarin.sru.server.config.SRUServerConfig attribute*), 20
response_NS (*clarin.sru.server.server.SRUNamespaces attribute*), 37
RESPONSE_POSITION (*clarin.sru.constants.SRUParam attribute*), 10
RESPONSE_POSITION (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
RESPONSE_POSITION_OUT_OF_RANGE (*clarin.sru.constants.SRUDiagnostics attribute*), 10
response_prefix (*clarin.sru.server.server.SRUNamespaces attribute*), 37
RESPONSE_TYPE (*clarin.sru.constants.SRUParam attribute*), 10
RESPONSE_TYPE (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
restrictions (*clarin.sru.server.config.DatabaseInfo attribute*), 19
RESULT_SET_CREATED_WITH_UNPREDICTABLE_PARTIAL_RESULTS_AVAILABLE (*clarin.sru.constants.SRUDiagnostics attribute*), 8
RESULT_SET_CREATED_WITH_VALID_PARTIAL_RESULTS_AVAILABLE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
RESULT_SET_DOES_NOT_EXIST (*clarin.sru.constants.SRUDiagnostics attribute*), 8
RESULT_SET_NOT_CREATED_TOO_MANY_MATCHING_RECORDS (*clarin.sru.constants.SRUDiagnostics attribute*), 9
RESULT_SET_TEMPORARILY_UNAVAILABLE (*clarin.sru.constants.SRUDiagnostics attribute*), 8
RESULT_SET_TTL (*clarin.sru.constants.SRUParam attribute*), 10
RESULT_SET_TTL (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
RESULT_SETS_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 8
RESULT_SETS_ONLY_SUPPORTED_FOR_RETRIEVAL (*clarin.sru.constants.SRUDiagnostics attribute*), 8
retrieve (*clarin.sru.server.config.SchemaInfo attribute*), 19

S

SCAN (*clarin.sru.constants.SRUOperation attribute*), 5
SCAN (*clarin.sru.server.request.ParameterInfoSets attribute*), 27
scan() (*clarin.sru.server.server.SRUSearchEngine method*), 38
scan() (*clarin.sru.server.server.SRUServer method*), 39
SCAN_CLAUSE (*clarin.sru.constants.SRUParam attribute*), 10
SCAN_CLAUSE (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
scan_NS (*clarin.sru.server.server.SRUNamespaces attribute*), 37
scan_prefix (*clarin.sru.server.server.SRUNamespaces attribute*), 37
schema_info (*clarin.sru.server.config.SRUServerConfig attribute*), 21
SchemaInfo (*class in clarin.sru.server.config*), 19
search() (*clarin.sru.server.server.SRUSearchEngine method*), 38
search() (*clarin.sru.server.server.SRUServer method*), 39
SEARCH_RETRIEVE (*clarin.sru.constants.SRUOperation attribute*), 5
SEARCH_RETRIEVE (*clarin.sru.server.request.ParameterInfoSets attribute*), 27
SEARCH_TERMS (*clarin.sru.constants.SRUQueryType attribute*), 5
SearchTermsQuery (*class in clarin.sru.queryparser*), 14

SearchTermsQueryParser (*class in clarin.sru.queryparser*), 14
SEEN_DATA (*clarin.sru.xml.writer.SRUXMLStreamWriter.IndentingState attribute*), 40
SEEN_ELEMENT (*clarin.sru.xml.writer.SRUXMLStreamWriter.IndentingState attribute*), 40
SEEN_NOTHING (*clarin.sru.xml.writer.SRUXMLStreamWriter.IndentingState attribute*), 40
SERVER (*clarin.sru.constants.SRURenderBy attribute*), 6
set (*clarin.sru.server.config.IndexInfo.Index.Map attribute*), 19
set_default_params() (*clarin.sru.server.wsgi.SRUServerApp method*), 39
setDocumentLocator() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40
setDocumentLocator() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 42
sets (*clarin.sru.server.config.IndexInfo attribute*), 20
skippedEntity() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 42
skippedEntity() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
sort (*clarin.sru.server.config.SchemaInfo attribute*), 19
SORT_ENDED_DUE_TO_MISSING_VALUE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
SORT_KEYS (*clarin.sru.constants.SRUParam attribute*), 10
SORT_KEYS (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
SORT_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 9
SORT_SPEC_INCLUDED_BOTH_IN_QUERY_AND_PROTOCOL_ERROR (*clarin.sru.constants.SRUDiagnostics attribute*), 9
SORT_SPEC_INCLUDED_BOTH_IN_QUERY_AND_PROTOCOL_PREVAILS (*clarin.sru.constants.SRUDiagnostics attribute*), 9
SORT_SPEC_INCLUDED_BOTH_IN_QUERY_AND_PROTOCOL_QUERY_PREVAILS (*clarin.sru.constants.SRUDiagnostics attribute*), 9
SRU_ALLOW_OVERRIDE_INDENT_RESPONSE (*clarin.sru.server.config.SRUServerConfigKey attribute*), 18
SRU_ALLOW_OVERRIDE_MAXIMUM_RECORDS (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_ALLOW_OVERRIDE_MAXIMUM_TERMS (*clarin.sru.server.config.SRUServerConfigKey attribute*), 18
SRU_DATABASE (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_ECHO_REQUESTS (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_HOST (*clarin.sru.server.config.SRUServerConfigKey attribute*), 16
SRU_INDENT_RESPONSE (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_LEGACY_NAMESPACE_MODE (*clarin.sru.server.config.SRUServerConfigKey attribute*), 16
SRU_MAXIMUM_RECORDS (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_MAXIMUM_TERMS (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_NUMBER_OF_RECORDS (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_NUMBER_OF_TERMS (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_PORT (*clarin.sru.server.config.SRUServerConfigKey attribute*), 17
SRU_RESPONSE_BUFFER_SIZE (*clarin.sru.server.config.SRUServerConfigKey attribute*), 18
SRU_SUPPORTED_VERSION_DEFAULT (*clarin.sru.server.config.SRUServerConfigKey attribute*), 16
SRU_SUPPORTED_VERSION_MAX (*clarin.sru.server.config.SRUServerConfigKey attribute*), 16
SRU_SUPPORTED_VERSION_MIN (*clarin.sru.server.config.SRUServerConfigKey attribute*), 16
SRU_TRANSPORT (*clarin.sru.server.config.SRUServerConfigKey attribute*), 16
SRUAbstractResult (*class in clarin.sru.server.result*), 32
SRUAuthenticationInfo (*class in clarin.sru.server.auth*), 15
SRUAuthenticationInfoProvider (*class in clarin.sru.server.auth*), 15
SRUAuthenticationInfoProviderFactory (*class in clarin.sru.server.auth*), 15
SRUConfigException, 12
SRUDiagnostic (*class in clarin.sru.diagnostic*), 11
SRUDiagnosticList (*class in clarin.sru.diagnostic*), 11
SRUDiagnostics (*class in clarin.sru.constants*), 6
SRUException, 12
SRUExplainResult (*class in clarin.sru.server.result*), 32
SRUNamespaces (*class in clarin.sru.server.server*), 37
SRUOperation (*class in clarin.sru.constants*), 5

SRUParam (*class in clarin.sru.constants*), 10
SRUParamValue (*class in clarin.sru.constants*), 10
SRUQuery (*class in clarin.sru.queryparser*), 12
SRUQueryParser (*class in clarin.sru.queryparser*), 12
SRUQueryParserRegistry (*class in clarin.sru.queryparser*), 13
SRUQueryParserRegistry.Builder (*class in clarin.sru.queryparser*), 13
SRUQueryType (*class in clarin.sru.constants*), 5
SRURecordPacking (*class in clarin.sru.constants*), 5
SRURecordXmlEscaping (*class in clarin.sru.constants*), 5
SRURenderBy (*class in clarin.sru.constants*), 6
SRURequest (*class in clarin.sru.server.request*), 22
SRURequestImpl (*class in clarin.sru.server.request*), 27
SRUResultCountPrecision (*class in clarin.sru.constants*), 6
SRUScanResultSet (*class in clarin.sru.server.result*), 32
SRUScanResultSet.WhereInList (*class in clarin.sru.server.result*), 33
SRUSearchEngine (*class in clarin.sru.server.server*), 37
SRUSearchResultSet (*class in clarin.sru.server.result*), 34
SRUServer (*class in clarin.sru.server.server*), 39
SRUServerApp (*class in clarin.sru.server.wsgi*), 39
SRUServerConfig (*class in clarin.sru.server.config*), 20
SRUServerConfigKey (*class in clarin.sru.server.config*), 16
SRUVersion (*class in clarin.sru.constants*), 6
SRUXMLStreamWriter (*class in clarin.sru.xml.writer*), 40
SRUXMLStreamWriter.IndentingState (*class in clarin.sru.xml.writer*), 40
START_RECORD (*clarin.sru.constants.SRUParam attribute*), 10
START_RECORD (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
startDocument() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
startDocument() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
startElement() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
startElement() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
startElementNS() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 41
startElementNS() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 43
startPrefixMapping() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40
startPrefixMapping() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 42
startRecord() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 40
startRecord() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
STRING (*clarin.sru.constants.SRURecordXmlEscaping attribute*), 5
STYLESHEET (*clarin.sru.constants.SRUParam attribute*), 10
STYLESHEET (*clarin.sru.server.request.ParameterInfo.Parameter attribute*), 26
STYLESHEETS_NOT_SUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 10
subject (*clarin.sru.server.auth.SRUAuthenticationInfo property*), 15
subjects (*clarin.sru.server.config.DatabaseInfo attribute*), 19
supports_version() (*clarin.sru.queryparser.CQLQueryParser method*), 15
supports_version() (*clarin.sru.queryparser.SearchTermsQueryParser method*), 14
supports_version() (*clarin.sru.queryparser.SRUQueryParser method*), 13
SYSTEM_TEMPORARILY_UNAVAILABLE (*clarin.sru.constants.SRUDiagnostics attribute*), 7

T

TEMP_OUTPUT_BUFFERING (*clarin.sru.server.server.SRUServer attribute*), 39
TERM_CONTAINS_ONLY_STOPWORDS (*clarin.sru.constants.SRUDiagnostics attribute*), 8
TERM_IN_INVALID_FORMAT_FOR_INDEX_OR_RELATION (*clarin.sru.constants.SRUDiagnostics attribute*), 8
title (*clarin.sru.server.config.DatabaseInfo attribute*), 18
title (*clarin.sru.server.config.IndexInfo.Index attribute*), 20
title (*clarin.sru.server.config.IndexInfo.Set attribute*), 19
title (*clarin.sru.server.config.SchemaInfo attribute*), 19
TOO_MANY_BOOLEAN_OPERATORS_IN_QUERY (*clarin.sru.constants.SRUDiagnostics attribute*), 8
TOO_MANY_CHARACTERS_IN_QUERY (*clarin.sru.constants.SRUDiagnostics attribute*), 7
TOO_MANY_CHARACTERS_IN_TERM (*clarin.sru.constants.SRUDiagnostics attribute*), 7

TOO_MANY_MASKING_CHARACTERS_IN_TERM (*clarin.sru.constants.SRUDiagnostics attribute*), 8
TOO_MANY_RECORDS_TO_SORT (*clarin.sru.constants.SRUDiagnostics attribute*), 9
TOO_MANY_SORT_KEYS_TO_SORT (*clarin.sru.constants.SRUDiagnostics attribute*), 9
TOO_MANY_TERMS_REQUESTED (*clarin.sru.constants.SRUDiagnostics attribute*), 10
transport (*clarin.sru.server.config.SRUServerConfig attribute*), 20

U

UNABLE_TO_EVALUATE_XPATH_EXPRESSION (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNKNOWN (*clarin.sru.constants.SRUResultCountPrecision attribute*), 6
UNKNOWN_SCHEMA_FOR_RETRIEVAL (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNPACKED (*clarin.sru.constants.SRURecordPacking attribute*), 5
UNSUPPORTED_BOOLEAN_MODIFIER (*clarin.sru.constants.SRUDiagnostics attribute*), 8
UNSUPPORTED_BOOLEAN_OPERATOR (*clarin.sru.constants.SRUDiagnostics attribute*), 8
UNSUPPORTED_CASE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNSUPPORTED_COMBINATION_OF_INDEXES (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_COMBINATION_OF_PROXIMITY_MODIFIERS (*clarin.sru.constants.SRUDiagnostics attribute*), 8
UNSUPPORTED_COMBINATION_OF_RELATION_AND_INDEX (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_COMBINATION_OF_RELATION_AND_TERM (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_COMBINATION_OF_RELATION_MODIFERS (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_CONTEXT_SET (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_DIRECTION (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNSUPPORTED_INDEX (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_MISSING_VALUE_ACTION (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNSUPPORTED_OPERATION (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_PARAMETER (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_PARAMETER_VALUE (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_PATH_FOR_SORT (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNSUPPORTED_PROXIMITY_DISTANCE (*clarin.sru.constants.SRUDiagnostics attribute*), 8
UNSUPPORTED_PROXIMITY_ORDERING (*clarin.sru.constants.SRUDiagnostics attribute*), 8
UNSUPPORTED_PROXIMITY_RELATION (*clarin.sru.constants.SRUDiagnostics attribute*), 8
UNSUPPORTED_PROXIMITY_UNIT (*clarin.sru.constants.SRUDiagnostics attribute*), 8
UNSUPPORTED_RELATION (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_RELATION_MODIFIER (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_SCHEMA_FOR_SORT (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNSUPPORTED_SORT_SEQUENCE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
UNSUPPORTED_STYLESHEET (*clarin.sru.constants.SRUDiagnostics attribute*), 10
UNSUPPORTED_VERSION (*clarin.sru.constants.SRUDiagnostics attribute*), 7
UNSUPPORTED_XML_ESCAPING_VALUE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
uri (*clarin.sru.diagnostic.SRUDiagnostic attribute*), 11

V

value (*clarin.sru.server.config.LocalizedString attribute*), 16
VERSION (*clarin.sru.constants.SRUParam attribute*), 10
VERSION_1_1 (*clarin.sru.constants.SRUParamValue attribute*), 11
VERSION_1_1 (*clarin.sru.constants.SRUVersion attribute*), 6
VERSION_1_2 (*clarin.sru.constants.SRUParamValue attribute*), 11
VERSION_1_2 (*clarin.sru.constants.SRUVersion attribute*), 6
VERSION_2_0 (*clarin.sru.constants.SRUVersion attribute*), 6
version_number (*clarin.sru.constants.SRUVersion property*), 6
version_string (*clarin.sru.constants.SRUVersion property*), 6

W

write_extra_record_data() (*clarin.sru.server.result.SRUSearchResultSet method*), 36
write_extra_response_data() (*clarin.sru.server.result.SRUAbstractResult method*), 32
write_extra_term_data() (*clarin.sru.server.result.SRUScanResultSet method*), 34
write_record() (*clarin.sru.server.result.SRUSearchResultSet method*), 36
writeXCQL() (*clarin.sru.xml.writer.SRUXMLStreamWriter method*), 42
writeXML() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
writeXMLdocument() (*clarin.sru.xml.writer.XMLStreamWriterHelper method*), 44
wsgi_app() (*clarin.sru.server.wsgi.SRUServerApp method*), 40

X

X_INDENT_RESPONSE (*clarin.sru.constants.SRUParam attribute*), 10
X_UNLIMITED_RESULTSET (*clarin.sru.constants.SRUParam attribute*), 10
X_UNLIMITED_TERMLIST (*clarin.sru.constants.SRUParam attribute*), 10
XCQL_NS (*clarin.sru.server.server.SRUNamespaces attribute*), 37
XML (*clarin.sru.constants.SRURecordXmlEscaping attribute*), 5
XMLStreamWriterHelper (*class in clarin.sru.xml.writer*), 42
XPATH_EXPRESSION_CONTAINS_UNSUPPORTED_FEATURE (*clarin.sru.constants.SRUDiagnostics attribute*), 9
XPATH_RETRIEVAL_UNSUPPORTED (*clarin.sru.constants.SRUDiagnostics attribute*), 9