elementpath Manual

Release 2.5.2

Davide Brunato

May 17, 2022

Contents

CHAPTER 1

Introduction

The proposal of this package is to provide XPath 1.0, 2.0 and 3.0 selectors for ElementTree XML data structures, both for the standard ElementTree library and for the lxml.etree library.

For lxml.etree this package can be useful for providing XPath 2.0/3.0 selectors, because lxml.etree already has it's own implementation of XPath 1.0.

1.1 Installation and usage

You can install the package with *pip* in a Python 3.7+ environment:

```
pip install elementpath
```

For using it import the package and apply the selectors on ElementTree nodes:

```
>>> import elementpath
>>> from xml.etree import ElementTree
>>> root = ElementTree.XML('<A><B1/><B2><C1/><C2/><C3/></B2></A>')
>>> elementpath.select(root, '/A/B2/*')
[<Element 'C1' at ...>, <Element 'C2' at ...>, <Element 'C3' at ...>]
```

The *select* API provides the standard XPath result format that is a list or an elementary datatype's value. If you want only to iterate over results you can use the generator function *iter_select* that accepts the same arguments of *select*.

The selectors API works also using XML data trees based on the lxml.etree library:

```
>>> import elementpath
>>> import lxml.etree as etree
>>> root = etree.XML('<A><B1/><B2><C1/><C2/><C3/></B2></A>')
>>> elementpath.select(root, '/A/B2/*')
[<Element C1 at ...>, <Element C2 at ...>, <Element C3 at ...>]
```

When you need to apply the same XPath expression to several XML data you can also use the *Selector* class, creating an instance and then using it to apply the path on distinct XML data:

```
>>> import elementpath
>>> import lxml.etree as etree
>>> selector = elementpath.Selector('/A/*/*')
>>> root = etree.XML('<A><B1/><B2><C1/><C2/><C3/></B2></A>')
>>> selector.select(root)
[<Element C1 at ...>, <Element C2 at ...>, <Element C3 at ...>]
>>> root = etree.XML('<A><B1><C0/></B1><B2><C1/><C2/><C3/></B2></A>')
>>> selector.select(root)
[<Element C0 at ...>, <Element C1 at ...>, <Element C2 at ...>, <Element C3 at ...>]
```

Public API classes and functions are described into the elementpath manual on the "Read the Docs" site.

For default the XPath 2.0 is used. If you need XPath 1.0 parser provide the *parser* argument:

```
>>> from elementpath import select, XPath1Parser
>>> from xml.etree import ElementTree
>>> root = ElementTree.XML('<A><B1/><B2><C1/><C2/><C3/></B2></A>')
>>> select(root, '/A/B2/*', parser=XPath1Parser)
[<Element 'C1' at ...>, <Element 'C2' at ...>, <Element 'C3' at ...>]
```

For XPath 3.0 import the parser from *elementpath.xpath3* subpackage, that is not loaded for default:

```
>>> from elementpath.xpath3 import XPath3Parser
>>> select(root, 'math:atan(1.0e0)', parser=XPath3Parser)
0.7853981633974483
```

1.2 Contributing

You can contribute to this package reporting bugs, using the issue tracker or by a pull request. In case you open an issue please try to provide a test or test data for reproducing the wrong behaviour. The provided testing code shall be added to the tests of the package.

The XPath parsers are based on an implementation of the Pratt's Top Down Operator Precedence parser. The implemented parser includes some lookup-ahead features, helpers for registering tokens and for extending language implementations. Also the token class has been generalized using a *MutableSequence* as base class. See *tdop_parser.py* for the basic internal classes and *xpath1_parser.py* for extensions and for a basic usage of the parser.

If you like you can use the basic parser and tokens provided by the *tdop_parser.py* module to implement other types of parsers (I think it could be also a funny exercise!).

1.3 License

CHAPTER 2

Public XPath API

The package includes some classes and functions that implement XPath selectors, parsers, tokens, contexts and schema proxy.

2.1 XPath selectors

select(*root: Any*, *path: str*, *namespaces: Optional[MutableMapping[str, str]] = None*, *parser: Optional[elementpath.xpath2.xpath2_parser.XPath2Parser] = None*, ***kwargs*) → Any

XPath selector function that apply a *path* expression on *root* Element.

Parameters

- **root** – an Element or ElementTree instance.
- **path** – the XPath expression.
- **namespaces** – a dictionary with mapping from namespace prefixes into URIs.
- **parser** – the parser class to use, that is *XPath2Parser* for default.
- **kwargs** – other optional parameters for the parser instance or the dynamic context.

Returns a list with XPath nodes or a basic type for expressions based on a function or literal.

iter_select(*root: Any*, *path: str*, *namespaces: Optional[MutableMapping[str, str]] = None*, *parser: Optional[elementpath.xpath2.xpath2_parser.XPath2Parser] = None*, ***kwargs*) → Iterator[Any]

A function that creates an XPath selector generator for apply a *path* expression on *root* Element.

Parameters

- **root** – an Element or ElementTree instance.
- **path** – the XPath expression.
- **namespaces** – a dictionary with mapping from namespace prefixes into URIs.
- **parser** – the parser class to use, that is *XPath2Parser* for default.

- **kwargs** – other optional parameters for the parser instance or the dynamic context.

Returns a generator of the XPath expression results.

class Selector(*path: str*, *namespaces: Optional[MutableMapping[str, str]] = None*, *parser: Optional[elementpath.xpath2.xpath2_parser.XPath2Parser] = None*, ***kwargs*)

XPath selector class. Create an instance of this class if you want to apply an XPath selector to several target data.

Parameters

- **path** – the XPath expression.
- **namespaces** – a dictionary with mapping from namespace prefixes into URIs.
- **parser** – the parser class to use, that is *XPath2Parser* for default.
- **kwargs** – other optional parameters for the XPath parser instance.

Variables

- **path** (*str*) – the XPath expression.
- **parser** (*XPath1Parser or XPath2Parser*) – the parser instance.
- **root_token** (*XPathToken*) – the root of tokens tree compiled from path.

namespaces

A dictionary with mapping from namespace prefixes into URIs.

select(*root: Any*, ***kwargs*) → Any

Applies the instance's XPath expression on *root* Element.

Parameters

- **root** – an Element or ElementTree instance.
- **kwargs** – other optional parameters for the XPath dynamic context.

Returns a list with XPath nodes or a basic type for expressions based on a function or literal.

iter_select(*root: Any*, ***kwargs*) → Iterator[Any]

Creates an XPath selector generator for apply the instance's XPath expression on *root* Element.

Parameters

- **root** – an Element or ElementTree instance.
- **kwargs** – other optional parameters for the XPath dynamic context.

Returns a generator of the XPath expression results.

2.2 XPath parsers

class XPath1Parser(*namespaces: Optional[MutableMapping[str, str]] = None*, *strict: bool = True*, **args*, ***kwargs*)

XPath 1.0 expression parser class. Provide a *namespaces* dictionary argument for mapping namespace prefixes to URI inside expressions. If *strict* is set to *False* the parser enables also the parsing of QNames, like the ElementPath library.

Parameters

- **namespaces** – a dictionary with mapping from namespace prefixes into URIs.

- **strict** – a strict mode is *False* the parser enables parsing of QNames in extended format, like the Python's ElementPath library. Default is *True*.

DEFAULT_NAMESPACES = {'xml': 'http://www.w3.org/XML/1998/namespace'}
The default prefix-to-namespace associations of the XPath class. These namespaces are updated in the instance with the ones passed with the *namespaces* argument.

version = '1.0'
The XPath version string.

default_namespace
The default namespace. For XPath 1.0 this value is always *None* because the default namespace is ignored (see https://www.w3.org/TR/1999/REC-xpath-19991116/#node-tests).

Helper methods for defining token classes:

classmethod axis(*symbol: str*, *reverse_axis: bool = False*, *bp: int = 80*) → Type[elementpath.xpath_token.XPathAxis]
Register a token for a symbol that represents an XPath *axis*.

classmethod function(*symbol: str, nargs: Union[int, Tuple[int, Optional[int]], None] = None, sequence_types: Tuple[str, ...] = (), label: str = 'function', bp: int = 90*) → Type[elementpath.xpath_token.XPathFunction]
Registers a token class for a symbol that represents an XPath function.

class XPath2Parser(*namespaces: Optional[MutableMapping[str, str]] = None*, *variable_types: Optional[Dict[str, str]] = None*, *strict: bool = True*, *compatibility_mode: bool = False*, *default_collation: Optional[str] = None*, *default_namespace: Optional[str] = None*, *function_namespace: Optional[str] = None*, *xsd_version: Optional[str] = None*, *schema: Optional[elementpath.schema_proxy.AbstractSchemaProxy] = None*, *base_uri: Optional[str] = None*, *document_types: Optional[Dict[str, str]] = None*, *collection_types: Optional[Dict[str, str]] = None*, *default_collection_type: str = 'node()*'*)
XPath 2.0 expression parser class. This is the default parser used by XPath selectors. A parser instance represents also the XPath static context. With *variable_types* you can pass a dictionary with the types of the in-scope variables. Provide a *namespaces* dictionary argument for mapping namespace prefixes to URI inside expressions. If *strict* is set to *False* the parser enables also the parsing of QNames, like the ElementPath library. There are some additional XPath 2.0 related arguments.

Parameters

- **namespaces** – a dictionary with mapping from namespace prefixes into URIs.
- **variable_types** – a dictionary with the static context's in-scope variable types. It defines the associations between variables and static types.
- **strict** – if strict mode is *False* the parser enables parsing of QNames, like the ElementPath library. Default is *True*.
- **compatibility_mode** – if set to *True* the parser instance works with XPath 1.0 compatibility rules.
- **default_namespace** – the default namespace to apply to unprefixed names. For default no namespace is applied (empty namespace '').
- **function_namespace** – the default namespace to apply to unprefixed function names. For default the namespace "http://www.w3.org/2005/xpath-functions" is used.
- **schema** – the schema proxy class or instance to use for types, attributes and elements lookups. If an *AbstractSchemaProxy* subclass is provided then a schema proxy instance is built without the optional argument, that involves a mapping of only XSD builtin types. If it's not provided the XPath 2.0 schema's related expressions cannot be used.

- **base_uri** – an absolute URI maybe provided, used when necessary in the resolution of relative URIs.
- **default_collation** – the default string collation to use. If not set the environment's default locale setting is used.
- **document_types** – statically known documents, that is a dictionary from absolute URIs onto types. Used for type check when calling the *fn:doc* function with a sequence of URIs. The default type of a document is 'document-node()'.
- **collection_types** – statically known collections, that is a dictionary from absolute URIs onto types. Used for type check when calling the *fn:collection* function with a sequence of URIs. The default type of a collection is 'node()*'.
- **default_collection_type** – this is the type of the sequence of nodes that would result from calling the *fn:collection* function with no arguments. Default is 'node()*'.

class XPath30Parser(**args, decimal_formats: Optional[Dict[Optional[str], Dict[str, str]]] = None, **kwargs*)

XPath 3.0 expression parser class. Accepts all XPath 2.0 options as keyword arguments, but the *strict* option is ignored because XPath 3.0+ has braced URI literals and the expanded name syntax is not compatible.

Parameters

- **args** – the same positional arguments of class `XPath2Parser`.
- **decimal_formats** – a mapping with statically known decimal formats.
- **kwargs** – the same keyword arguments of class `XPath2Parser`.

2.3 XPath tokens

class XPathToken(*parser: elementpath.tdop.Parser[elementpath.tdop.Token[~TK]][elementpath.tdop.Token[~TK][TK]], value: Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None] = None*)

Base class for XPath tokens.

evaluate(*context: Optional[elementpath.xpath_context.XPathContext] = None*) → Any

Evaluate default method for XPath tokens.

Parameters **context** – The XPath dynamic context.

select(*context: Optional[elementpath.xpath_context.XPathContext] = None*) → Iterator[Any]

Select operator that generates XPath results.

Parameters **context** – The XPath dynamic context.

Context manipulation helpers:

get_argument(*context: Optional[elementpath.xpath_context.XPathContext], index: int = 0, required: bool = False, default_to_context: bool = False, default: Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None] = None, cls: Optional[Type[Any]] = None, promote: Union[Type[Any], Tuple[Type[Any], ...], None] = None*) → Any

Get the argument value of a function of constructor token. A zero length sequence is converted to a *None* value. If the function has no argument returns the context's item if the dynamic context is not *None*.

Parameters

- **context** – the dynamic context.
- **index** – an index for select the argument to be got, the first for default.
- **required** – if set to *True* missing or empty sequence arguments are not allowed.
- **default_to_context** – if set to *True* then the item of the dynamic context is returned when the argument is missing.
- **default** – the default value returned in case the argument is an empty sequence. If not provided returns *None*.
- **cls** – if a type is provided performs a type checking on item.
- **promote** – a class or a tuple of classes that are promoted to *cls* class.

atomization(*context: Optional[elementpath.xpath_context.XPathContext] = None*) → Iterator[Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth]]

Helper method for value atomization of a sequence.

Ref: https://www.w3.org/TR/xpath20/#id-atomization

Parameters **context** – the XPath dynamic context.

get_atomized_operand(*context: Optional[elementpath.xpath_context.XPathContext] = None*) → Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None]

Get the atomized value for an XPath operator.

Parameters **context** – the XPath dynamic context.

Returns the atomized value of a single length sequence or *None* if the sequence is empty.

iter_comparison_data(*context: elementpath.xpath_context.XPathContext*) → Iterator[Tuple[Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None], Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None]]]

Generates comparison data couples for the general comparison of sequences. Different sequences maybe generated with an XPath 2.0 parser, depending on compatibility mode setting.

Ref: https://www.w3.org/TR/xpath20/#id-general-comparisons

Parameters **context** – the XPath dynamic context.

get_operands(*context: elementpath.xpath_context.XPathContext*, *cls: Optional[Type[Any]] = None*) → Tuple[Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None], Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None]]

Returns the operands for a binary operator. Float arguments are converted to decimal if the other argument is a *Decimal* instance.

Parameters

- **context** – the XPath dynamic context.
- **cls** – if a type is provided performs a type checking on item.

Returns a couple of values representing the operands. If any operand is not available returns a *(None, None)* couple.

get_results(*context: elementpath.xpath_context.XPathContext*) → Union[List[Any], str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth]

Returns formatted XPath results.

Parameters **context** – the XPath dynamic context.

Returns a list or a simple datatype when the result is a single simple type generated by a literal or function token.

select_results(*context: Optional[elementpath.xpath_context.XPathContext]*) → Iterator[Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, Any, Tuple[str, str]]]

Generates formatted XPath results.

Parameters **context** – the XPath dynamic context.

adjust_datetime(*context: elementpath.xpath_context.XPathContext, cls: Type[Union[elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth]]*) → Union[elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, elementpath.datatypes.datetime.DayTimeDuration, None]

XSD datetime adjust function helper.

Parameters

- **context** – the XPath dynamic context.
- **cls** – the XSD datetime subclass to use.

Returns an empty list if there is only one argument that is the empty sequence or the adjusted XSD datetime instance.

use_locale(*collation: str*) → Iterator[None]

A context manager for use a locale setting for string comparison in a code block.

Schema context methods .. automethod:: select_xsd_nodes .. automethod:: add_xsd_type .. automethod:: get_xsd_type .. automethod:: get_typed_node

Data accessor helpers .. automethod:: data_value .. automethod:: boolean_value .. automethod:: string_value .. automethod:: number_value .. automethod:: schema_node_value

Error management helper:

error(*code: Union[str, elementpath.datatypes.qname.QName], message_or_error: Union[None, str, Exception] = None*) → elementpath.exceptions.ElementPathError

Returns an XPath error instance related with a code. An XPath/XQuery/XSLT error code is an alphanumeric token starting with four uppercase letters and ending with four digits.

Parameters

- **code** – the error code as QName or string.
- **message_or_error** – an optional custom message or an exception.

2.4 XPath contexts

class XPathContext(*root: Any*, *namespaces: Optional[Dict[str, str]] = None*, *item: Union[Any, elementpath.xpath_nodes.XPathNode, elementpath.datatypes.atomic_types.AnyAtomicType, None] = None*, *position: int = 1*, *size: int = 1*, *axis: Optional[str] = None*, *variables: Optional[Dict[str, Any]] = None*, *current_dt: Optional[datetime.datetime] = None*, *timezone: Union[str, elementpath.datatypes.datetime.Timezone, None] = None*, *documents: Optional[Dict[str, Any]] = None*, *collections: Optional[Dict[str, Any]] = None*, *default_collection: Optional[str] = None*, *text_resources: Optional[Dict[str, str]] = None*, *resource_collections: Optional[Dict[str, List[str]]] = None*, *default_resource_collection: Optional[str] = None*, *allow_environment: bool = False*, *default_language: Optional[str] = None*, *default_calendar: Optional[str] = None*, *default_place: Optional[str] = None*)

The XPath dynamic context. The static context is provided by the parser.

Usually the dynamic context instances are created providing only the root element. Variable values argument is needed if the XPath expression refers to in-scope variables. The other optional arguments are needed only if a specific position on the context is required, but have to be used with the knowledge of what is their meaning.

Parameters

- **root** – the root of the XML document, can be a ElementTree instance or an Element.
- **namespaces** – a dictionary with mapping from namespace prefixes into URIs, used when namespace information is not available within document and element nodes. This can be useful when the dynamic context has additional namespaces and root is an Element or an ElementTree instance of the standard library.
- **item** – the context item. A *None* value means that the context is positioned on the document node.
- **position** – the current position of the node within the input sequence.
- **size** – the number of items in the input sequence.
- **axis** – the active axis. Used to choose when apply the default axis ('child' axis).
- **variables** – dictionary of context variables that maps a QName to a value.
- **current_dt** – current dateTime of the implementation, including explicit timezone.
- **timezone** – implicit timezone to be used when a date, time, or dateTime value does not have a timezone.
- **documents** – available documents. This is a mapping of absolute URI strings onto document nodes. Used by the function fn:doc.
- **collections** – available collections. This is a mapping of absolute URI strings onto sequences of nodes. Used by the XPath 2.0+ function fn:collection.
- **default_collection** – this is the sequence of nodes used when fn:collection is called with no arguments.
- **text_resources** – available text resources. This is a mapping of absolute URI strings onto text resources. Used by XPath 3.0+ function fn:unparsed-text/fn:unparsed-text-lines.
- **resource_collections** – available URI collections. This is a mapping of absolute URI strings to sequence of URIs. Used by the XPath 3.0+ function fn:uri-collection.
- **default_resource_collection** – this is the sequence of URIs used when fn:uri-collection is called with no arguments.

- **allow_environment** – defines if the access to system environment is allowed, for default is *False*. Used by the XPath 3.0+ functions fn:environment-variable and fn:available-environment-variables.

class XPathSchemaContext(*root: Any*, *namespaces: Optional[Dict[str, str]] = None*, *item: Union[Any, elementpath.xpath_nodes.XPathNode, elementpath.datatypes.atomic_types.AnyAtomicType, None] = None*, *position: int = 1*, *size: int = 1*, *axis: Optional[str] = None*, *variables: Optional[Dict[str, Any]] = None*, *current_dt: Optional[datetime.datetime] = None*, *timezone: Union[str, elementpath.datatypes.datetime.Timezone, None] = None*, *documents: Optional[Dict[str, Any]] = None*, *collections: Optional[Dict[str, Any]] = None*, *default_collection: Optional[str] = None*, *text_resources: Optional[Dict[str, str]] = None*, *resource_collections: Optional[Dict[str, List[str]]] = None*, *default_resource_collection: Optional[str] = None*, *allow_environment: bool = False*, *default_language: Optional[str] = None*, *default_calendar: Optional[str] = None*, *default_place: Optional[str] = None*)

The XPath dynamic context base class for schema bounded parsers. Use this class as dynamic context for schema instances in order to perform a schema-based type checking during the static analysis phase. Don't use this as dynamic context on XML instances.

2.5 XML Schema proxy

The XPath 2.0 parser can be interfaced with an XML Schema processor through a schema proxy. An `XMLSchemaProxy` class is defined for interfacing schemas created with the *xmlschema* package. This class is based on an abstract class `AbstractSchemaProxy`, that can be used for implementing concrete interfaces to other types of XML Schema processors.

class AbstractSchemaProxy(*schema: Any*, *base_element: Optional[Any] = None*)

Abstract base class for defining schema proxies.

Parameters

- **schema** – a schema instance that implements the *AbstractEtreeElement* interface.
- **base_element** – the schema element used as base item for static analysis.

bind_parser(*parser: Any*) → None

Binds a parser instance with schema proxy adding the schema's atomic types constructors. This method can be redefined in a concrete proxy to optimize schema bindings.

Parameters **parser** – a parser instance.

get_context() → elementpath.xpath_context.XPathSchemaContext

Get a context instance for static analysis phase.

Returns an *XPathSchemaContext* instance.

find(*path: str*, *namespaces: Optional[Dict[str, str]] = None*) → Optional[Any]

Find a schema element or attribute using an XPath expression.

Parameters

- **path** – an XPath expression that selects an element or an attribute node.
- **namespaces** – an optional mapping from namespace prefix to namespace URI.

Returns The first matching schema component, or `None` if there is no match.

get_type(*qname: str*) → Optional[Any]
Get the XSD global type from the schema's scope. A concrete implementation must return an object that supports the protocols *XsdTypeProtocol*, or *None* if the global type is not found.

Parameters **qname** – the fully qualified name of the type to retrieve.

Returns an object that represents an XSD type or *None*.

get_attribute(*qname: str*) → Optional[Any]
Get the XSD global attribute from the schema's scope. A concrete implementation must return an object that supports the protocol *XsdAttributeProtocol*, or *None* if the global attribute is not found.

Parameters **qname** – the fully qualified name of the attribute to retrieve.

Returns an object that represents an XSD attribute or *None*.

get_element(*qname: str*) → Optional[Any]
Get the XSD global element from the schema's scope. A concrete implementation must return an object that supports the protocol *XsdElementProtocol* interface, or *None* if the global element is not found.

Parameters **qname** – the fully qualified name of the element to retrieve.

Returns an object that represents an XSD element or *None*.

is_instance(*obj: Any*, *type_qname: str*) → bool
Returns *True* if *obj* is an instance of the XSD global type, *False* if not.

Parameters

- **obj** – the instance to be tested.
- **type_qname** – the fully qualified name of the type used to test the instance.

cast_as(*obj: Any*, *type_qname: str*) → Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth]
Converts *obj* to the Python type associated with an XSD global type. A concrete implementation must raises a *ValueError* or *TypeError* in case of a decoding error or a *KeyError* if the type is not bound to the schema's scope.

Parameters

- **obj** – the instance to be casted.
- **type_qname** – the fully qualified name of the type used to convert the instance.

iter_atomic_types() → Iterator[Any]
Returns an iterator for not builtin atomic types defined in the schema's scope. A concrete implementation must yields objects that implement the protocol *XsdTypeProtocol*.

get_primitive_type(*xsd_type: Any*) → Any
Returns the type at base of the definition of an XSD type. For an atomic type is effectively the primitive

type. For a list is the primitive type of the item. For a union is the base union type. For a complex type is xs:anyType.

Parameters **xsd_type** – an XSD type instance.

Returns an XSD type instance.

2.6 XPath nodes

XPath nodes are processed using a set of classes derived from `XPathNode`:

class AttributeNode(*name: str, value: Union[str, Any], parent: Optional[Any] = None*)
A class for processing XPath attribute nodes.

Parameters

- **name** – the attribute name.
- **value** – a string value or an XSD attribute when XPath is applied on a schema.
- **parent** – the parent element.

class TextNode(*value: str*, *parent: Optional[Any] = None*, *tail: bool = False*)
A class for processing XPath text nodes. An Element's property (elem.text or elem.tail) with a *None* value is not a text node.

Parameters

- **value** – a string value.
- **parent** – the parent element.
- **tail** – provide *True* if the text node is the parent Element's tail.

class TypedAttribute(*attribute: elementpath.xpath_nodes.AttributeNode*, *xsd_type: Any*, *value: Any*)
A class for processing typed attribute nodes.

Parameters

- **attribute** – the origin AttributeNode instance.
- **xsd_type** – the reference XSD type.
- **value** – the types value.

class TypedElement(*elem: Any*, *xsd_type: Any*, *value: Any*)
A class for processing typed element nodes.

Parameters

- **elem** – the linked element. Can be an Element, or an XSD element when XPath is applied on a schema.
- **xsd_type** – the reference XSD type.
- **value** – the decoded value. Can be *None* for empty or element-only elements."

class NamespaceNode(*prefix: str*, *uri: str*, *parent: Optional[Any] = None*)
A class for processing XPath namespace nodes.

Parameters

- **prefix** – the namespace prefix.
- **uri** – the namespace URI.

- **parent** – the parent element.

2.7 XPath regular expressions

translate_pattern(*pattern: str*, *flags: int = 0*, *xsd_version: str = '1.0'*, *back_references: bool = True*, *lazy_quantifiers: bool = True*, *anchors: bool = True*) → str

Translates a pattern regex expression to a Python regex pattern. With default options the translator processes XPath 2.0/XQuery 1.0 regex patterns. For XML Schema patterns set all boolean options to *False*.

Parameters

- **pattern** – the source XML Schema regular expression.
- **flags** – regex flags as represented by Python's re module.
- **xsd_version** – apply regex rules of a specific XSD version, '1.0' for default.
- **back_references** – if *True* supports back-references and capturing groups.
- **lazy_quantifiers** – if *True* supports lazy quantifiers (*?, +?).
- **anchors** – if *True* supports ^ and $ anchors, otherwise the translated pattern is anchored to its boundaries and anchors are treated as normal characters.

2.8 Exception classes

exception ElementPathError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

Base exception class for elementpath package.

Parameters

- **message** – the message related to the error.
- **code** – an optional error code.
- **token** – an optional token instance related with the error.

exception MissingContextError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

Raised when the dynamic context is required for evaluate the XPath expression.

exception RegexError

Error in a regular expression or in a character class specification. This exception is derived from *Exception* base class and is raised only by the regex subpackage.

There are also other exceptions, multiple derived from the base exception `ElementPathError` and Python built-in exceptions:

exception ElementPathKeyError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

exception ElementPathLocaleError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

exception ElementPathNameError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

exception ElementPathOverflowError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

exception ElementPathSyntaxError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

exception ElementPathTypeError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

exception ElementPathValueError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

exception ElementPathZeroDivisionError(*message: str*, *code: Optional[str] = None*, *token: Optional[elementpath.tdop.Token[typing.Any][Any]] = None*)

CHAPTER 3

Pratt's parser API

The TDOP (Top Down Operator Precedence) parser implemented within this library is a variant of the original Pratt's parser based on a class for the parser and meta-classes for tokens.

The parser base class includes helper functions for registering token classes, the Pratt's methods and a regexp-based tokenizer builder. There are also additional methods and attributes to help the developing of new parsers. Parsers can be defined by class derivation and following a tokens registration procedure. These classes are not available at package level but only within module *elementpath.tdop*.

3.1 Token base class

class Token(*parser: elementpath.tdop.Parser[elementpath.tdop.Token[~TK]][elementpath.tdop.Token[~TK][TK]], value: Union[str, int, float, decimal.Decimal, bool, elementpath.datatypes.numeric.Integer, elementpath.datatypes.numeric.Float10, elementpath.datatypes.string.NormalizedString, elementpath.datatypes.uri.AnyURI, elementpath.datatypes.binary.HexBinary, elementpath.datatypes.binary.Base64Binary, elementpath.datatypes.qname.QName, elementpath.datatypes.datetime.AbstractDateTime, elementpath.datatypes.datetime.Duration, elementpath.datatypes.untyped.UntypedAtomic, elementpath.datatypes.datetime.OrderedDateTime, elementpath.datatypes.datetime.Date10, elementpath.datatypes.datetime.Date, elementpath.datatypes.datetime.DateTime10, elementpath.datatypes.datetime.DateTime, elementpath.datatypes.datetime.Time, elementpath.datatypes.datetime.GregorianDay, elementpath.datatypes.datetime.GregorianMonth, elementpath.datatypes.datetime.GregorianMonthDay, elementpath.datatypes.datetime.GregorianYear10, elementpath.datatypes.datetime.GregorianYear, elementpath.datatypes.datetime.GregorianYearMonth10, elementpath.datatypes.datetime.GregorianYearMonth, None] = None*)

Token base class for defining a parser based on Pratt's method.

Each token instance is a list-like object. The number of token's items is the arity of the represented operator, where token's items are the operands. Nullary operators are used for symbols, names and literals. Tokens with items represent the other operators (unary, binary and so on).

Each token class has a *symbol*, a lbp (left binding power) value and a rbp (right binding power) value, that are used in the sense described by the Pratt's method. This implementation of Pratt tokens includes two extra attributes, *pattern* and *label*, that can be used to simplify the parsing of symbols in a concrete parser.

Parameters

- **parser** – The parser instance that creates the token instance.
- **value** – The token value. If not provided defaults to token symbol.

Variables

- **symbol** – the symbol of the token class.
- **lbp** – Pratt's left binding power, defaults to 0.
- **rbp** – Pratt's right binding power, defaults to 0.
- **pattern** – the regex pattern used for the token class. Defaults to the escaped symbol. Can be customized to match more detailed conditions (eg. a function with its left round bracket), in order to simplify the related code.
- **label** – defines the typology of the token class. Its value is used in representations of the token instance and can be used to restrict code choices without more complicated analysis. The label value can be set as needed by the parser implementation (eg. 'function', 'axis', 'constructor function' are used by the XPath parsers). In the base parser class defaults to 'symbol' with 'literal' and 'operator' as possible alternatives. If set by a tuple of values the token class label is transformed to a multi-value label, that means the token class can covers multiple roles (eg. as XPath function or axis). In those cases the definitive role is defined at parse time (nud and/or led methods) after the token instance creation.

arity

tree
Returns a tree representation string.

source
Returns the source representation string.

nud() → TK
Pratt's null denotation method

led(*left: TK*) → TK
Pratt's left denotation method

evaluate() → Any
Evaluation method

iter(**symbols*) → Iterator[elementpath.tdop.Token[~TK][TK]]
Returns a generator for iterating the token's tree.

Helper methods for checking symbols and for error raising:

expected(**symbols*, *message: Optional[str] = None*) → None

unexpected(**symbols*, *message: Optional[str] = None*) → None

wrong_syntax(*message: Optional[str] = None*) → elementpath.tdop.ParseError

wrong_value(*message: str = 'invalid value'*) → ValueError

wrong_type(*message: str = 'invalid type'*) → TypeError

3.2 Parser base class

class Parser

Parser class for implementing a Top Down Operator Precedence parser.

Variables

- **SYMBOLS** – the symbols of the definable tokens for the parser. In the base class it's an immutable set that contains the symbols for special tokens (literals, names and end-token). Has to be extended in a concrete parser adding all the symbols of the language.
- **symbol_table** – a dictionary that stores the token classes defined for the language.
- **token_base_class** – the base class for creating language's token classes.
- **tokenizer** – the language tokenizer compiled regexp.

position

Property that returns the current line and column indexes.

Parsing methods:

parse(*source: str*) → TK_co

Parses a source code of the formal language. This is the main method that has to be called for a parser's instance.

Parameters **source** – The source string.

Returns The root of the token's tree that parse the source.

advance(**symbols*) → TK_co

The Pratt's function for advancing to next token.

Parameters **symbols** – Optional arguments tuple. If not empty one of the provided symbols is expected. If the next token's symbol differs the parser raises a parse error.

Returns The next token instance.

advance_until(**stop_symbols*) → str

Advances until one of the symbols is found or the end of source is reached, returning the raw source string placed before. Useful for raw parsing of comments and references enclosed between specific symbols.

Parameters **stop_symbols** – The symbols that have to be found for stopping advance.

Returns The source string chunk enclosed between the initial position and the first stop symbol.

expression(*rbp: int = 0*) → TK_co

Pratt's function for parsing an expression. It calls token.nud() and then advances until the right binding power is less the left binding power of the next token, invoking the led() method on the following token.

Parameters **rbp** – right binding power for the expression.

Returns left token.

Helper methods for checking parser status:

is_source_start() → bool

Returns *True* if the parser is positioned at the start of the source, ignoring the spaces.

is_line_start() → bool

Returns *True* if the parser is positioned at the start of a source line, ignoring the spaces.

is_spaced(*before: bool = True*, *after: bool = True*) → bool

Returns *True* if the source has an extra space (whitespace, tab or newline) immediately before or after the current position of the parser.

Parameters

- **before** – if *True* considers also the extra spaces before the current token symbol.
- **after** – if *True* considers also the extra spaces after the current token symbol.

Helper methods for building new parsers:

classmethod register(*symbol: Union[str, Type[TK_co]], **kwargs*) → Type[TK_co]
Register/update a token class in the symbol table.

Parameters

- **symbol** – The identifier symbol for a new class or an existent token class.
- **kwargs** – Optional attributes/methods for the token class.

Returns A token class.

classmethod unregister(*symbol: str*) → None
Unregister a token class from the symbol table.

classmethod duplicate(*symbol: str*, *new_symbol: str*, ***kwargs*) → Type[TK_co]
Duplicate a token class with a new symbol.

classmethod literal(*symbol: str*, *bp: int = 0*) → Type[TK_co]
Register a token for a symbol that represents a *literal*.

classmethod nullary(*symbol: str*, *bp: int = 0*) → Type[TK_co]
Register a token for a symbol that represents a *nullary* operator.

classmethod prefix(*symbol: str*, *bp: int = 0*) → Type[TK_co]
Register a token for a symbol that represents a *prefix* unary operator.

classmethod postfix(*symbol: str*, *bp: int = 0*) → Type[TK_co]
Register a token for a symbol that represents a *postfix* unary operator.

classmethod infix(*symbol: str*, *bp: int = 0*) → Type[TK_co]
Register a token for a symbol that represents an *infix* binary operator.

classmethod infixr(*symbol: str*, *bp: int = 0*) → Type[TK_co]
Register a token for a symbol that represents an *infixr* binary operator.

classmethod method(*symbol: Union[str, Type[TK_co]], bp: int = 0*) → Callable[[Callable[[...], Any]], Callable[[...], Any]]
Register a token for a symbol that represents a custom operator or redefine a method for an existing token.

classmethod build() → None
Builds the parser class. Checks if all declared symbols are defined and builds a the regex tokenizer using the symbol related patterns.

classmethod create_tokenizer(*symbol_table: MutableMapping[str, Type[TK_co]]*) → Pattern[str]
Returns a regex based tokenizer built from a symbol table of token classes. The returned tokenizer skips extra spaces between symbols.

A regular expression is created from the symbol table of the parser using a template. The symbols are inserted in the template putting the longer symbols first. Symbols and their patterns can't contain spaces.

Parameters **symbol_table** – a dictionary containing the token classes of the formal language.

Index