OpyenXes Documentation

Release 0.3.0

Process Mining UC

Sep 18, 2018

Contents

Contents:

CHAPTER 1

OpyenXES

A python implementation of the XES standard based on the Java implementation OpenXes.

- Free software: GNU General Public License v3
- Documentation: https://opyenxes.readthedocs.io.

1.1 Citing OpyenXES

The package OpyenXES is an academic project. Therefore, the time and resources spent to develop OpyenXES are justified by the number of citations made to the corresponding publication. If you publish scientific works that make use of OpyenXES, please cite the following paper (bibtex):

> Valdivieso, H., Lee, W. L. J., Munoz-Gama, J., & Sepúlveda, M. OpyenXES: A Complete Python Library for the eXtensible Event Stream Standard.

1.2 Features

- Facilitate io with XES, MXML event log files
- Implementation of the XES standard
- Based on the well-used Java implementation OpenXes

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

1.4 Development Lead

- Hernan Valdivieso (The guy who wrote everything!)
- Jorge Munoz-Gama (The mastermind)
- Wai Lam Jonathan Lee

1.5 Contributors

- TKasekamp

CHAPTER 2

Installation

2.1 Stable release

To install OpyenXes, run this command in your terminal:

```
$ pip install opyenxes
```

This is the preferred method to install OpyenXes, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for OpyenXes can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/opyenxes/OpyenXes
```

Or download the tarball:

```
$ curl  -OL https://github.com/opyenxes/OpyenXes/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use OpyenXes in a project:

```
import opyenxes
```

CHAPTER 4

src

4.1 opyenxes package

4.1.1 Subpackages

opyenxes.classification package

Submodules

opyenxes.classification.XEventAndClassifier module

class opyenxes.classification.XEventAndClassifier.**XEventAndClassifier**(*comparators*)

Bases: *opyenxes.classification.XEventAttributeClassifier.XEventAttributeClassifier*

Composite event classifier, which can hold any number of lower-level classifiers, concatenated with boolean AND logic. This classifier will consider two events as equal, if all of its lower-level classifiers consider them as equal.

Parameters **comparators** (list[*XEventAttributeClassifier*]) – Any number of lower-level classifiers, which are evaluated with boolean AND logic. If multiple lower-level classifiers use the same keys, this key is used only once in this classifier.

opyenxes.classification.XEventAttributeClassifier module

class opyenxes.classification.XEventAttributeClassifier.**XEventAttributeClassifier**(*name*, *keys*)

Bases: object

Event classifier which considers two events as equal, if, for a set of given (configurable) attributes, they have the same values.

Parameters

- **name** (*str*) – Name of the classifier.
- **keys** (*list[str]*) – List with the keys of the attributes used for event comparison.

compare_to(*obj*)
Helper method to compares this object with the specified object for order.

Parameters **obj** (*XAttributeDiscrete*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_class_identity(*event*)
Retrieves the unique class identity string of a given event.

Parameters **event** (*XEvent*) – The given event to transform in the identity string.

Returns The string that represent that event with this classifier.

Return type str

get_defining_attribute_keys()
Retrieves the set of attribute keys which are used in this event classifier (May be used for the construction of events that are not part of an existing event class).

Returns An array of attribute keys, which are used for defining this classifier.

Return type list[str]

name()
Returns the name of this comparator

Returns The name of this comparator.

Return type str

same_event_class(*event_a*, *event_b*)
Checks whether two event instances correspond to the same event class, i.e. are equal in that sense.

Parameters

- **event_a** (*XEvent*) – The first event to check.
- **event_b** (*XEvent*) – The second event to check with the first.

Returns True if two event have the same event class, False otherwise.

Return type bool

set_name(*name*)
Assigns a custom name to this classifier

Parameters **name** – Name to be assigned to this classifier.

Return type str

opyenxes.classification.XEventClass module

class opyenxes.classification.XEventClass.**XEventClass**(*identity*, *index*)
Bases: object

Implements an event class. An event class is an identity for events, making them comparable. If two events are part of the same class, they are considered to be equal, i.e. to be referring to the same higher-level concept.

Parameters

- **identity** (*str*) – Unique identification string of the class, i.e. its name.
- **index** (*int*) – Unique index of class.

compare_to(*obj*)
Helper method to compares this object with the specified object for order.

Parameters **obj** (*XAttributeDiscrete*) – The Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_id()

Retrieves the name, i.e. unique identification string, of this event class.

Returns The name of this class, as a unique string.

Return type str

get_index()
Returns the index of this event class.

Returns Unique index.

Return type int

increment_size()
Increments the size of this class by one, i.e. adds another event to the number of represented instances.

set_size(*size*)
Sets the size of this event class, i.e. the number of represented instances.

Parameters **size** (*int*) – Number of events in this class.

size()

Retrieves the size, i.e. the number of events represented by this event class.

Returns Size of this class.

Return type int

opyenxes.classification.XEventClasses module

class opyenxes.classification.XEventClasses.**XEventClasses**(*classifier*)
Bases: object

A set of event classes. For any log, this class can be used to impose a classification of events. Two events which belong to the same event class can be considered equal, i.e. to refer to the same higher-level concept they represent (e.g., an activity). Event classes are imposed on a log by a specific classifier. This class can be configured with such a classifier, which is then used to derive the actual event classes from a log, by determining the identity of the contained events.

Parameters **classifier** (*XEventAttributeClassifier*) – The classifier used for creating the set of event classes.

static derive_event_classes(*classifier*, *log*)
Creates a new set of event classes, factory method.

Parameters

- **classifier** (*XEventAttributeClassifier*) – The classifier to be used for event comparison.
- **log** (*XLog*) – The log, on which event classes should be imposed.

Returns A set of event classes, as an instance of this class.

Return type *XEventClasses*

get_by_identity(*identity*)
Returns a given event class by its identity, i.e. its unique identifier string.

Parameters **identity** (*str*) – Identifier string of the requested event class.

Returns The requested event class. If no matching event class is found, this method may return null.

Return type *XEventClass*

get_by_index(*index*)
Returns a given event class by its unique index.

Parameters **index** (*int*) – Unique index of the requested event class.

Returns The requested event class. If no matching event class is found, this method may return null.

Return type *XEventClass*

get_class_of(*event*)
For any given event, returns the corresponding event class as determined by this set.

Parameters **event** (*XEvent*) – The event of which the event class should be determined.

Returns

The event class of this event, as found in this set of event classes. If no matching event class is found, this method may return

null.

Return type *XEventClass*

get_classes()
Returns the collection of event classes contained in this instance.

Returns A collection of event classes.

Return type dict_values

get_classifier()
Returns the classifier used for determining event classes.

Returns A classifier used in this set of classes.

Return type dict

harmonize_indices()
This method harmonizeds the indices of all contained event classes. Indices are re-assigned according to the natural order of class identities, i.e., the alphabetical order of class identity strings. This method should be called after the composition or derivation of event classes is complete, e.g., after scanning a log for

generating the log info. Using parties should not have to worry about event class harmonization, and can thus safely ignore this method.

register (*element*)
Registers a XES Element(log, trace and event and class ID) with this set of event classes. This will result in all events of this log being analyzed, and potentially new event classes being added to this set of event classes. Event classes will be incremented in size, as new members of these classes are found among the events in the log.

Parameters **element** (*XLog* or *XTrace* or *XEvent* or str) – The Xes Element or Class ID to be analyzed.

opyenxes.classification.XEventLifeTransClassifier module

class opyenxes.classification.XEventLifeTransClassifier.**XEventLifeTransClassifier**
Bases: *opyenxes.classification.XEventAttributeClassifier.XEventAttributeClassifier*

Implements an event classifier based on the lifecycle transition attribute of events.

opyenxes.classification.XEventNameClassifier module

class opyenxes.classification.XEventNameClassifier.**XEventNameClassifier**
Bases: *opyenxes.classification.XEventAttributeClassifier.XEventAttributeClassifier*

Implements an event classifier based on the activity name of events.

opyenxes.classification.XEventResourceClassifier module

class opyenxes.classification.XEventResourceClassifier.**XEventResourceClassifier**
Bases: *opyenxes.classification.XEventAttributeClassifier.XEventAttributeClassifier*

Implements an event classifier based on the resource name attribute of events.

Module contents

opyenxes.data_in package

Submodules

opyenxes.data_in.XMxmlGZIPParser module

class opyenxes.data_in.XMxmlGZIPParser.**XMxmlGZIPParser** (*factory=None*)
Bases: *opyenxes.data_in.XMxmlParser.XMxmlParser*

Parser for the compressed MXML format for event logs (deprecated).

Parameters **factory** (*XFactory*) – The factory to use for XES model building.

can_parse (*file*)
Checks whether this parser can handle the given file.

Parameters **file** (*str*) – path of the file to check against parser.

Returns Whether this parser can handle the given file.

Return type bool

parse(*file*)

Parses a log from the given input stream, which is supposed to deliver an XES log in XML representation.

Parameters **file** (*_io.TextIOWrapper*) – file generated by the function 'open(path)', which is supposed to deliver an XES log in XML representation.

Returns The parsed log.

Return type list[*XLog*]

opyenxes.data_in.XMxmlParser module

class opyenxes.data_in.XMxmlParser.**XMxmlParser**(*factory=None*)

Bases: object

Parser for the MXML format for event logs (deprecated).

Parameters **factory** (*XFactory*) – The factory to use for XES model building.

MXML_CLASSIFIERS = [<opyenxes.classification.XEventAttributeClassifier.XEventAttribute

class MxmlHandler

Bases: xml.sax.handler.ContentHandler

SAX handler class for XES in XML representation.

characters(*content*)

Overrides characters in class ContentHandler

Parameters **content** (*str*) – The characters.

endElement(*local_name*)

Overrides endElement in class ContentHandler

Parameters **local_name** (*str*) – The name of the element type, just as with the startElement event

get_logs()

Retrieves the parsed list of logs.

Returns The parsed list of logs.

Return type list[*XLog*]

ignorableWhitespace(*whitespace*)

Overrides ignorableWhitespace in class ContentHandler

Parameters **whitespace** (*str*) – The whitespace characters.

startElement(*element_name*, *attributes*)

Overrides startElement in class ContentHandler

Parameters

- **element_name** (*str*) – Contains the raw XML 1.0 name of the element type.
- **attributes** (*xml.sax.xmlreader.AttributesImpl*) – An instance of the Attributes class containing the attributes of the element

can_parse(*file*)

Checks whether this parser can handle the given file.

Parameters **file** (*str*) – path of the file to check against parser.

Returns Whether this parser can handle the given file.

Return type bool

static ends_with_ignore_case(*name*, *suffix*)
Returns whether the given file name ends (ignoring the case) with the given suffix.

Parameters

- **name** (*str*) – The given file name.
- **suffix** (*str*) – The given suffix.

Returns Whether the given file name ends (ignoring the case) with the given suffix.

Return type bool

parse(*file*)
Parses a set of logs from the given input stream, which is supposed to deliver an MXML serialization.

Parameters **file** (*_io.TextIOWrapper*) – file generated by the function 'open(path)', which is supposed to deliver an MXML serialization.

Returns The parsed list of logs.

Return type list[*XLog*]

opyenxes.data_in.XParserRegistry module

class opyenxes.data_in.XParserRegistry.**XParserRegistry**
Bases: *opyenxes.utils.XRegistry.XRegistry*

System-wide registry for XES parser implementations. Applications can use this registry as a convenience to provide an overview about parseable formats, e.g., in the user interface. Any custom parser implementation can be registered with this registry, so that it transparently becomes available also to any other using application.

Uses the singleton metaclass

opyenxes.data_in.XUniversalParser module

class opyenxes.data_in.XUniversalParser.**XUniversalParser**
Bases: object

This class implements a universal parser, using the parser registry to find an appropriate parser for extracting an XES model from any given file. May be used as a convenience method for applications.

static can_parse(*file*)
Checks whether the given file can be parsed by any parser.

Parameters **file** (*str*) – path of the file to check against parser.

Returns Whether this parser can handle the given file.

Return type bool

static parse(*file*)
Attempts to parse a collection of XES models from the given file, using all available parsers.

Parameters **file** (*_io.TextIOWrapper*) – file generated by the function 'open(path)', which is supposed to deliver an XES log in XML representation.

Returns The parsed list of logs.

Return type list[*XLog*]

opyenxes.data_in.XesXmlGZIPParser module

class opyenxes.data_in.XesXmlGZIPParser.**XesXmlGZIPParser**(*factory=None*)
Bases: *opyenxes.data_in.XesXmlParser.XesXmlParser*

Parser for the compressed XES XML serialization.

Parameters **factory** (*XFactory*) – The XES model factory instance used to build the model from the serialization.

can_parse(*file*)
Checks whether this parser can handle the given file.

Parameters **file** (*str*) – path of the file to check against parser.

Returns Whether this parser can handle the given file.

Return type bool

parse(*file*)
Parses a log from the given input stream, which is supposed to deliver an XES log in XML representation.

Parameters **file** (*_io.TextIOWrapper*) – file generated by the function 'open(path)', which is supposed to deliver an XES log in XML representation.

Returns The parsed list of logs.

Return type list[*XLog*]

opyenxes.data_in.XesXmlParser module

class opyenxes.data_in.XesXmlParser.**XesXmlParser**(*factory=None*)
Bases: object

Parser for the XES XML serialization.

Parameters **factory** (*XFactory*) – The XES model factory instance used to build the model from the serialization.

class XesXmlHandler
Bases: xml.sax.handler.ContentHandler

SAX handler class for XES in XML representation.

endElement(*local_name*)
Overrides endElement in class ContentHandler
Parameters **local_name** (*str*) – The name of the element type, just as with the startElement event

get_log()
Retrieves the parsed log.
Returns The parsed log.
Return type *XLog*

startElement(*element_name*, *attributes*)
Overrides startElement in class ContentHandler
Parameters
- **element_name** (*str*) – Contains the raw XML 1.0 name of the element type.

- **attributes** (*xml.sax.xmlreader.AttributesImpl*) – An instance of the Attributes class containing the attributes of the element

can_parse(*file*)
Checks whether this parser can handle the given file.

Parameters **file** (*str*) – path of the file to check against parser.

Returns Whether this parser can handle the given file.

Return type bool

static ends_with_ignore_case(*name*, *suffix*)
Returns whether the given file name ends (ignoring the case) with the given suffix.

Parameters

- **name** (*str*) – The given file name.
- **suffix** (*str*) – The given suffix.

Returns Whether the given file name ends (ignoring the case) with the given suffix.

Return type bool

parse(*file*)
Parses a log from the given input stream, which is supposed to deliver an XES log in XML representation.

Parameters **file** (*_io.TextIOWrapper*) – file generated by the function 'open(path)', which is supposed to deliver an XES log in XML representation.

Returns The parsed list of logs.

Return type list[*XLog*]

Module contents

opyenxes.data_out package

Submodules

opyenxes.data_out.XMxmlGZIPSerializer module

class opyenxes.data_out.XMxmlGZIPSerializer.**XMxmlGZIPSerializer**
Bases: *opyenxes.data_out.XMxmlSerializer.XMxmlSerializer*

Compressed MXML serialization for XES data (legacy implementation). Note that this serialization may be lossy, you should preferrably use the XES.XML serialization for XES data.

static get_suffices()
Returns an array of possible file suffices for this serialization.

Returns An array of possible file suffices for this serialization.

Return type list[str]

serialize(*log*, *out*, *in_bytes=False*)
Serializes a given log to the given output stream.

Parameters

- **log** (*XLog*) – Log to be serialized.

- **out** (*_io.TextIOWrapper*) – TextIOWrapper for serialization.
- **in_bytes** (*bool*) – Private argument to decide if serialized as bytes or as string

opyenxes.data_out.XMxmlSerializer module

class opyenxes.data_out.XMxmlSerializer.**XMxmlSerializer**
Bases: object

MXML serialization for XES data (legacy implementation). Note that this serialization may be lossy, you should preferrably use the XES.XML serialization for XES data.

add_attribute(*tag*, *attributes*, *key_prefix=None*)
Helper method, adds attributes to a tag.

Parameters

- **tag** (*xml.etree.ElementTree.Element*) – The tag to add attributes to.
- **attributes** (*dict_values*) – The attributes to add.
- **key_prefix** (*str*) – The Key prefix of attributes.

static add_model_reference(*element*, *target*)
Helper method, adds all model references of an attributable to the given tag.

Parameters

- **element** (*XAttributable*) – Attributable element.
- **target** (*xml.etree.ElementTree.Element*) – Tag to add model references to.

static get_suffices()
Returns an array of possible file suffices for this serialization.

Returns An array of possible file suffices for this serialization.

Return type list[str]

serialize(*log*, *out*, *in_bytes=False*)
Serializes a given log to the given output stream.

Parameters

- **log** (*XLog*) – Log to be serialized.
- **out** (*_io.TextIOWrapper*) – TextIOWrapper for serialization.
- **in_bytes** (*bool*) – Private argument to decide if serialized in bytes or in string

opyenxes.data_out.XSerializerRegistry module

class opyenxes.data_out.XSerializerRegistry.**XSerializerRegistry**
Bases: *opyenxes.utils.XRegistry.XRegistry*

System-wide registry for XES serializer implementations. Applications can use this registry as a convenience to provide an overview about serializeable formats, e.g., in the user interface. Any custom serializer implementation can be registered with this registry, so that it transparently becomes available also to any other using application.

Uses the singleton metaclass

opyenxes.data_out.XesXmlGZIPSerializer module

class opyenxes.data_out.XesXmlGZIPSerializer.**XesXmlGZIPSerializer**
Bases: *opyenxes.data_out.XesXmlSerializer.XesXmlSerializer*

XES compressed XML serialization for the XES format.

static get_suffices()
Returns an array of possible file suffices for this serialization.

Returns An array of possible file suffices for this serialization.

Return type list[str]

serialize(*log*, *out*, *in_bytes=False*)
Serializes a given log to the given output stream.

Parameters

- **log** (*XLog*) – Log to be serialized.
- **out** (*_io.TextIOWrapper*) – TextIOWrapper for serialization.
- **in_bytes** (*bool*) – Private argument to decide if serialized as bytes or as string

opyenxes.data_out.XesXmlSerializer module

class opyenxes.data_out.XesXmlSerializer.**XesXmlSerializer**
Bases: object

XES plain XML serialization for the XES format.

add_attributes(*tag*, *attributes*)
Helper method, adds the given collection of attributes to the given Tag.

Parameters

- **tag** (*xml.etree.ElementTree.Element*) – Tag to add attributes to.
- **attributes** (*dict_values*) – Collection with the attributes to add.

add_global_attributes(*parent*, *scope*, *attributes*)
Helper method for defining global attributes on a given scope.

Parameters

- **parent** (*xml.etree.ElementTree.Element*) – xml element to add a children element with the global attributes
- **scope** (*str*) – Name of the global attributes, can be 'trace' or 'event'.
- **attributes** (*dict_values*) – Collection with the attributes to add.

static get_author()
Returns the name of this serialization's author.

Returns The author name.

Return type str

static get_suffices()
Returns an array of possible file suffices for this serialization.

Returns An array of possible file suffices for this serialization.

Return type list[str]

serialize(*log*, *out*, *in_bytes=False*)
Serializes a given log to the given output stream.

Parameters

- **log** (*XLog*) – Log to be serialized.
- **out** (*_io.TextIOWrapper*) – TextIOWrapper for serialization.
- **in_bytes** (*bool*) – Private argument to decide if serialized as bytes or as string

Module contents

opyenxes.extension package

Subpackages

opyenxes.extension.std package

Submodules

opyenxes.extension.std.XAbstractNestedAttributeSupport module

class opyenxes.extension.std.XAbstractNestedAttributeSupport.**XAbstractNestedAttributeSuppo**
Bases: object

This class offers generic support for extracting and assigning values to and from nested attributes.

assign_nested_values(*element*, *amounts*)
Assigns (to the given event) multiple values given their key lists. The i-th element in the key list should correspond to an i-level attribute with the prescribed key. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key. For example, the call::

```
assignNestedValues(event, {["key.1"]: val.1, ["key.1", "key.1.1"]: val.1.1, [
↪"key.1", "key.1.2"]: val.1.2, ["key.2"]: val.2, ["key.3"]: val.3})
```

should result into the following XES fragment::

```
<event>
    <string key="key.1" value="">
        <float key="ext:attr" value="val.1"/>
        <string key="key.1.1" value="">
            <float key="ext:attr" value="val.1.1"/>
        </string>
        <string key="key.1.2" value="">
            <float key="ext:attr" value="val.1.2"/>
        </string>
    </string>
    <string key="key.2" value="">
       <float key="ext:attr" value="val.2"/>
    </string>
    <string key="key.3" value="">
       <float key="ext:attr" value="val.3"/>
    </string>
</event>
```

Parameters

- **element** (*XAttributable*) – Element to assign the values to.
- **amounts** (*dict(list[str]: Any)*) – Dictionary with key lists to values which are to be assigned.

classmethod assign_value(*element*, *value*)

Abstract method to assign a value to an element.

Parameters

- **element** (XAttribute) – The element to assign the value to.
- **value** (*Any*) – The value to be assigned.

assign_values(*element*, *values*)

Assigns (to the given element) multiple values given their keys. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key. For example, the call::

```
assign_values(event, {"key.1": val.1, "key.2": val.2, "key.3": val.3})
```

should result into the following XES fragment::

```
<event>
    <string key="key.1" value="">
        <float key="ext:attr" value="val.1"/>
    </string>
    <string key="key.2" value="">
       <float key="ext:attr" value="val.2"/>
    </string>
    <string key="key.3" value="">
       <float key="ext:attr" value="val.3"/>
    </string>
</event>
```

Parameters

- **element** (*XAttributable*) – Element to assign the values to.
- **values** (*dict(str: Any)*) – dictionary with keys to values which are to be assigned.

extract_nested_values(*element*)

Retrieves a map containing all values for all descending attributes of an element. For example, the XES fragment::

```
<trace>
    <string key="key.1" value="">
        <float key="ext:attr" value="val.1"/>
        <string key="key.1.1" value="">
            <float key="ext:attr" value="val.1.1"/>
        </string>
        <string key="key.1.2" value="">
            <float key="ext:attr" value="val.1.2"/>
        </string>
    </string>
    <string key="key.2" value="">
       <float key="ext:attr" value="val.2"/>
```

(continues on next page)

(continued from previous page)

```
    </string>
    <string key="key.3" value="">
       <float key="ext:attr" value="val.3"/>
    </string>
</trace>
```

should result into the following::

```
{["key.1"]: val.1, ["key.1", "key.1.1"]: val.1.1, ["key.1", "key.1.2"]: val.1.
↪2, ["key.2"]: val.2, ["key.3"]: val.3}
```

Parameters **element** (*XAttributable*) – Element to retrieve all values for.

Returns Dictionary with all descending keys to values.

Return type dict(list[str]: Any)

classmethod extract_value(*element*)

Abstract method to extract a value from an element.

Parameters **element** (*XAttribute*) – The element to extract the value from.

extract_values(*element*)

Retrieves a map containing all values for all child attributes of an element. For example, the XES fragment:

```
<trace>
    <string key="key.1" value="">
        <float key="ext:attr" value="val.1"/>
        <string key="key.1.1" value="">
            <float key="ext:attr" value="val.1.1"/>
        </string>
        <string key="key.1.2" value="">
            <float key="ext:attr" value="val.1.2"/>
        </string>
    </string>
    <string key="key.2" value="">
       <float key="ext:attr" value="val.2"/>
    </string>
    <string key="key.3" value="">
       <float key="ext:attr" value="val.3"/>
    </string>
</trace>
```

should result into the following::

```
{"key.1": val.1, "key.2": val.2, "key.3": val.3}
```

Parameters **element** (*XAttributable*) – Element to retrieve all values for.

Returns Dictionary with all child keys to values.

Return type dict(str: any}

opyenxes.extension.std.XConceptExtension module

class opyenxes.extension.std.XConceptExtension.**XConceptExtension**

Bases: *opyenxes.extension.XExtension.XExtension*

This extension provides naming for concepts in the event log type hierarchy. It defines two attributes:

- concept:name: Name (of any type hierarchy element)
- concept:instance: Instance identifier (of events)

Uses the singleton metaclass

assign_instance(*event*, *instance*)
Assigns any event its activity instance identifier, as defined by this extension's instance attribute.

Parameters

- **event** (*XEvent*) – Event to assign activity instance identifier to.
- **instance** (*str*) – The activity instance identifier to be assigned.

assign_name(*element*, *name*)
Assigns any log data hierarchy element its name, as defined by this extension's name attribute.

Parameters

- **element** (*XAttributable*) – Log hierarchy element to assign name to.
- **name** (*str*) – The name to be assigned.

static extract_instance(*event*)
Retrieves the activity instance identifier of an event, if set by this extension's instance attribute.

Parameters **event** (*XEvent*) – Event to extract instance from.

Returns The requested activity instance identifier.

Return type str

static extract_name(*element*)
Retrieves the name of a log data hierarchy element, if set by this extension's name attribute.

Parameters **element** (*XAttributable*) – Log hierarchy element to extract name from.

Returns The requested element name.

Return type str

opyenxes.extension.std.XCostExtension module

class opyenxes.extension.std.XCostExtension.**XCostAmount**
Bases: *opyenxes.extension.std.XAbstractNestedAttributeSupport.XAbstractNestedAttributeSupport*

Class which value contains the cost amount for a cost driver.

Uses the singleton metaclass

assign_value(*attribute*, *value*)
Abstract method to assign a value to an element.

Parameters

- **attribute** (*XAttribute*) – The element to assign the value to.
- **value** (*float*) – The value to be assigned.

extract_value(*attribute*)
Abstract method to extract a value from an element.

Parameters **attribute** (*XAttribute*) – The element to extract the value from.

Returns The extracted value.

Return type float

class opyenxes.extension.std.XCostExtension.**XCostDriver**
Bases: *opyenxes.extension.std.XAbstractNestedAttributeSupport.XAbstractNestedAttributeSupport*

Class which value contains the cost amount for a cost driver.

Uses the singleton metaclass

assign_value(*attribute*, *value*)
Abstract method to assign a value to an element.

Parameters

- **attribute** (*XAttribute*) – The element to assign the value to.
- **value** (*str*) – The value to be assigned.

extract_value(*attribute*)
Abstract method to extract a value from an element.

Parameters **attribute** (*XAttribute*) – The element to extract the value from.

Returns The extracted value.

Return type str

class opyenxes.extension.std.XCostExtension.**XCostExtension**
Bases: *opyenxes.extension.XExtension.XExtension*

This extension provides costs for traces and events. It defines five attributes:

- cost total: Contains total cost incurred for a trace or an event. The value represents the sum of all the cost amounts within the element.
- cost currecny: Any valid currency format.
- cost amount: The value contains the cost amount for a cost driver.
- cost driver: The value contains the id for the cost driver used to calculate the cost.
- cost type: The value contains the cost type (e.g., Fixed, Overhead, Materials).

Uses the singleton metaclass

assign_amount(*attribute*, *amount*)
Assigns any attribute its cost amount, as defined by this extension's amount attribute.

Parameters

- **attribute** (*XAttribute*) – Attribute to assign cost amount to
- **amount** (*float*) – The cost amount to be assigned.

static assign_amounts(*element*, *amounts*)
Assigns (to the given trace or event) multiple amounts given their keys. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key. For example, the call::

```
assign_amounts(trace, {"a": 10.00, "b": 15.00, "c": 25.00})
```

should result into the following XES fragment::

```
<trace>
    <string key="a" value="">
        <float key="cost:amount" value="10.00"/>
    </string>
    <string key="b" value="">
        <float key="cost:amount" value="15.00"/>
    </string>
    <string key="c" value="">
        <float key="cost:amount" value="25.00"/>
    </string>
</trace>
```

Parameters

- **element** (*XEvent* or *XTrace*) – Trace or Event to assign the amounts to.
- **amounts** (*dict(str: float)*) – Dictionary with keys to amounts which are to be assigned.

assign_currency(*element*, *currency*)
Assigns any trace or event its cost currency, as defined by this extension's currency attribute.

Parameters

- **element** (*XEvent* or *XEvent*) – Trace or Event to assign cost currency to.
- **currency** (*str*) – The currency to be assigned.

assign_driver(*attribute*, *driver*)
Assigns any attribute its cost driver, as defined by this extension's driver attribute.

Parameters

- **attribute** (*XAttribute*) – Attribute to assign cost driver to.
- **driver** (*str*) – The cost driver to be assigned.

static assign_drivers(*element*, *drivers*)
Assigns (to the given trace or event) multiple cost drivers given their keys. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key.

Parameters

- **element** (*XTrace* or *XEvent*) – Trace or Event to assign the cost drivers to.
- **drivers** (*dict(str: str)*) – Dictionary with keys to cost drivers which are to be assigned.

static assign_nested_amounts(*element*, *amounts*)
Assigns (to the given trace or event) multiple amounts given their key lists. The i-th element in the key list should correspond to an i-level attribute with the prescribed key. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key. For example, the call::

```
assign_nested_amounts(trace, {["a"]: 10.00], ["a", "b"]: 20.00, ["a", "c"]:␣
↪30.00, ["b"]: 15.00, ["c"]: 25.00})
```

should result into the following XES fragment::

```
<trace>
    <string key="a" value="">
        <float key="cost:amount" value="10.00"/>
```

(continues on next page)

(continued from previous page)

```
        <string key="b" value="">
            <float key="cost:amount" value="20.00"/>
        </string>
        <string key="c" value="">
            <float key="cost:amount" value="30.00"/>
        </string>
    </string>
    <string key="b" value="">
       <float key="cost:amount" value="15.00"/>
    </string>
    <string key="c" value="">
       <float key="cost:amount" value="25.00"/>
    </string>
</trace>
```

Parameters

- **element** (*XEvent* or *XTrace*) – Trace or Event to assign the amounts to.
- **amounts** (*dict(list[str]: float)*) – Dictionary with list of keys to amounts which are to be assigned.

static assign_nested_drivers(*element*, *drivers*)

Assigns (to the given trace) multiple cost drivers given their key lists. The i-th element in the key list should correspond to an i-level attribute with the prescribed key. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key.

Parameters

- **element** (*XTrace* or *XEvent*) – Trace or Event to assign the cost drivers to.
- **drivers** (*dict(list[str]: str)*) – Dictionary with keys to cost drivers which are to be assigned.

static assign_nested_types(*element*, *types*)

Assigns (to the given trace or event) multiple cost types given their key lists. The i-th element in the key list should correspond to an i-level attribute with the prescribed key. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key.

Parameters

- **element** (*XTrace* or *XEvent*) – Event or Trace to assign the cost types to.
- **types** (*dict(list[str]: str)*) – Dictionary with keys to cost types which are to be assigned.

assign_total(*element*, *total*)

Assigns any trace or event its total costs, as defined by this extension's total attribute.

Parameters

- **element** (*XTrace* or *XEvent*) – Trace or Event to assign total costs to.
- **total** (*float*) – The total costs to be assigned.

assign_type(*attribute*, *type_*)

Assigns any attribute its cost type, as defined by this extension's type attribute.

Parameters

- **attribute** (*XAttribute*) – Attribute to assign cost type to.

- **type** (*str*) – The cost type to be assigned.

static assign_types(*element*, *types*)

Assigns (to the given event or trace) multiple cost types given their keys. Note that as a side effect this method creates attributes when it does not find an attribute with the proper key.

Parameters

- **element** (*XTrace* or *XEvent*) – Event or Trace to assign the cost types to.
- **types** (*dict(list[str]: str)*) – Dictionary with keys to cost types which are to be assigned.

static extract_amount(*attribute*)

Retrieves the cost amount for an attribute, if set by this extension's amount attribute.

Parameters **attribute** (*XAttribute*) – Attribute element to retrieve cost amount for.

Returns The requested cost amount.

Return type float

static extract_amounts(*element*)

Retrieves a map containing all cost amounts for all child attributes of an event. For example, the XES fragment::

```
<trace>
    <string key="a" value="">
        <float key="cost:amount" value="10.00"/>
        <string key="b" value="">
            <float key="cost:amount" value="20.00"/>
        </string>
        <string key="c" value="">
            <float key="cost:amount" value="30.00"/>
        </string>
    </string>
    <string key="b" value="">
        <float key="cost:amount" value="15.00"/>
    </string>
    <string key="c" value="">
        <float key="cost:amount" value="25.00"/>
    </string>
</trace>
```

should result into the following::

```
{"a": 10.00, "b": 15.00, "c": 25.00}
```

Parameters **element** (*XTrace* or *XEvent*) – Trace or Event to retrieve all cost amounts for.

Returns Dictionary with all child keys to cost amounts.

Return type dict(str: float)

static extract_currency(*element*)

Retrieves the cost currency for an event or trace, if set by this extension's currency attribute.

Parameters **element** (*XTrace* or *XEvent*) – Event or Trace to retrieve currency for.

Returns The requested cost currency.

Return type str

static extract_driver(*attribute*)
Retrieves the cost driver for an attribute, if set by this extension's driver attribute.

Parameters **attribute** (*XAttribute*) – Attribute element to retrieve cost driver for.

Returns The requested cost driver.

Return type str

static extract_drivers(*element*)
Retrieves a map containing all cost drivers for all child attributes of a trace or event.

Parameters **element** (*XTrace* or *XEvent*) – Trace or Event to retrieve all cost drivers for.

Returns Dictionary with all child keys to cost drivers.

Return type dict(str: str)

static extract_nested_amounts(*element*)
Retrieves a map containing all cost amounts for all child attributes of an event. For example, the XES fragment::

```
<trace>
    <string key="a" value="">
        <float key="cost:amount" value="10.00"/>
        <string key="b" value="">
            <float key="cost:amount" value="20.00"/>
        </string>
        <string key="c" value="">
            <float key="cost:amount" value="30.00"/>
        </string>
    </string>
    <string key="b" value="">
        <float key="cost:amount" value="15.00"/>
    </string>
    <string key="c" value="">
        <float key="cost:amount" value="25.00"/>
    </string>
</trace>
```

should result into the following::

```
{["a"]: 10.00, ["a", "b"]: 20.00, ["a", "c"]: 30.00, ["b"]: 15.00, ["c"]: 25.
↪00}
```

Parameters **element** (*XTrace* or *XEvent*) – Trace or Event to retrieve all cost amounts for.

Returns Dictionary with all descending keys to cost amounts.

Return type dict(list[str]: float)

static extract_nested_drivers(*element*)
Retrieves a map containing all cost drivers for all descending attributes of a trace or event.

Parameters **element** (*XTrace* or *XEvent*) – Trace or Event to retrieve all cost drivers for.

Returns Dictionary with all descending keys to cost drivers.

Return type dict(list[str]: str)

static extract_nested_types(*element*)
Retrieves a map containing all cost types for all descending attributes of a trace or event.

Parameters **element** (*XTrace* or *XEvent*) – Trace or Event to retrieve all cost types for.

Returns Dictionary with all descending keys to cost types.

Return type dict(list[str]: str)

static extract_total(*element*)
Retrieves the total costs of a trace or event, if set by this extension's total attribute.

Parameters **element** (*XTrace* or *XEvent*) – Trace or Event to retrieve total costs for.

Returns The requested total costs.

Return type float

static extract_type(*attribute*)
Retrieves the cost type for an attribute, if set by this extension's type attribute.

Parameters **attribute** (*XAttribute*) – Attribute element to retrieve cost type for.

Returns The requested cost type.

Return type str

static extract_types(*element*)
Retrieves a map containing all cost types for all child attributes of a trace or event.

Parameters **element** (*XTrace* or *XEvent*) – Trace or Event to retrieve all cost types for.

Returns Dictionary with all child keys to cost types.

Return type dict(str: str)

class opyenxes.extension.std.XCostExtension.**XCostType**
Bases: *opyenxes.extension.std.XAbstractNestedAttributeSupport.XAbstractNestedAttributeSupport*

Class which value contains the cost type (e.g., Fixed,Overhead, Materials).

Uses the singleton metaclass

assign_value(*attribute*, *value*)
Abstract method to assign a value to an element.

Parameters

- **attribute** (*XAttribute*) – The element to assign the value to.
- **value** (*str*) – The value to be assigned.

extract_value(*attribute*)
Abstract method to extract a value from an element.

Parameters **attribute** (*XAttribute*) – The element to extract the value from.

Returns The extracted value.

Return type str

opyenxes.extension.std.XExtendedEvent module

class opyenxes.extension.std.XExtendedEvent.**XExtendedEvent**(*event*)
Bases: object

Helper class. This class can be used to dynamically wrap any event, and provides an extended set of getter and setter methods for typically-available extension attributes.

Parameters **event** (*XEvent*) – The original event to be wrapped.

clone()
Clones this event, i.e. creates a deep copy, but with a new ID, so equals does not hold between this and the clone

Returns An identical clone.

Return type *XExtendedEvent*

get_attributes()
Retrieves the attributes set for this element.

Returns A map of attributes.

Return type *XAttributeMap*

get_extensions()
Retrieves the extensions used by this element, i.e. the extensions used by all attributes of this element, and the element itself

Returns A set of extensions

Return type set(*XExtension*)

get_group()
Returns the group of the event, as defined by the Organizational extension.

Returns Group string. Can be null, if not defined.

Return type str

get_id()
Retrieves the id value of this event

Returns id of this event

Return type *XID*

get_instance()
Retrieves the activity instance of this event, as defined by the Concept extension.

Returns Activity instance of the event.

Return type str

get_model_references()
Returns the list of model references defined for this event, as defined in the Semantic extension.

Returns List of model reference strings.

Return type list[str]

get_model_references_uris()
Returns the list of model reference URIs defined for this event, as defined in the Semantic extension.

Returns List of model reference URIs.

Return type list[ParseResult]

get_name()
Retrieves the activity name of this event, as defined by the Concept extension.

Returns Activity name of the event.

Return type str

get_resourse()
Returns the resource of the event, as defined by the Organizational extension.

Returns Resource string. Can be null, if not defined.

Return type str

get_role()
Returns the role of the event, as defined by the Organizational extension.

Returns Role string. Can be null, if not defined.

Return type str

get_standard_transition()
Returns the standard lifecycle transition of the event, as defined by the Lifecycle extension.

Returns Standard lifecycle transition as string. Can be null, if not defined.

Return type str

get_time_stamp()
Retrieves the timestamp of the event, as defined by the Time extension.

Returns Timestamp as Date object, or null if not defined.

Return type datetime.datetime

get_transition()
Returns the lifecycle transition of the event, as defined by the Lifecycle extension.

Returns Lifecycle transition string. Can be null, if not defined.

Return type str

has_attributes()
Checks for the existence of attributes

Returns True if this element has any attributes; False otherwise.

Return type bool

set_attributes(*attributes*)
Sets the map of attributes for this element.

Parameters **attributes** (*XAttributeMap*) – A map of attributes

set_group(*group*)
Sets the group of the event, as defined by the Organizational extension.

Parameters **group** (*str*) – Group string.

set_instance(*instance*)
Sets the activity instance of this event, as defined by the Concept extension.

Parameters **instance** (*str*) – Activity instance of the event.

set_model_references(*model_references*)
Sets the list of model reference strings defined for this event, as defined in the Semantic extension.

Parameters **model_references** (*list[str]*) – List of model reference strings.

set_model_references_uris(*model_references_uris*)
Sets the list of model reference URIs defined for this event, as defined in the Semantic extension.

Parameters **model_references_uris** (*list[ParseResult or SplitResult]*) – List of model reference URIs.

set_name(*name*)
Sets the activity name of this event, as defined by the Concept extension.

Parameters **name** (*str*) – Activity instance of the event.

set_resource(*resource*)
Sets the resource of the event, as defined by the Organizational extension.

Parameters **resource** (*str*) – Resource string.

set_role(*role*)
Sets the role of the event, as defined by the Organizational extension.

Parameters **role** (*str*) – Role string.

set_standard_transition(*transition*)
Sets the standard lifecycle transition of the event, as defined by the Lifecycle extension.

Parameters **transition** (*str*) – Standard lifecycle transition object as string.

set_time_stamp(*time_stamp*)
Sets the timestamp of the event, as defined by the Time extension.

Parameters **time_stamp** – Timestamp, as Date or as long value in milliseconds,

to be set. :type time_stamp: datetime.datetime or int

set_transition(*transition*)
Sets the lifecycle transition of the event, as defined by the Lifecycle extension.

Parameters **transition** (*str*) – Lifecycle transition string.

static wrap(*event*)
Static wrapper method. Wraps the given event into an instance of this class, which transparently provides extended access to attributes.

Parameters **event** (*XEvent*) – The original event to be wrapped.

Returns A wrapped event.

Return type *XExtendedEvent*

opyenxes.extension.std.XIdentityExtension module

class opyenxes.extension.std.XIdentityExtension.**XIdentityExtension**
Bases: *opyenxes.extension.XExtension.XExtension*

Provides unique identifiers (UUIDs) for elements.

Uses the singleton metaclass

assign_id(*element*, *identity*)
Assigns any log data hierarchy element its id, as defined by this extension's id attribute.

Parameters

- **element** (*XAttributable*) – Log hierarchy element to assign id to.
- **identity** (XID) – The id to be assigned.

static extract_id(*element*)
Retrieves the id of a log data hierarchy element, if set by this extension's id attribute.

Parameters **element** (*XAttributable*) – Log hierarchy element to assign name from.

Returns The requested element id.

Return type *XID*

static extract_name(*element*)
Retrieves the name of a log data hierarchy element, if set by this extension's name attribute.

Parameters **element** (*XAttributable*) – Log hierarchy element to extract name from.

Returns The requested element name.

Return type str

opyenxes.extension.std.XLifecycleExtension module

class opyenxes.extension.std.XLifecycleExtension.**XLifecycleExtension**
Bases: *opyenxes.extension.XExtension.XExtension*

Extension defining additional attributes for the event lifecycle. Lifecycles define a set of states for activities, with an accompanying set of transitions between those states. Any event which is referring to by a lifecycle represents a certain transition of an activity within that lifecycle.

Uses the singleton metaclass

class StandardModel
Bases: object

Class with the standard lifecycle model.

ASSIGN = 'assign'

ATE_ABORT = 'ate_abort'

AUTOSKIP = 'autoskip'

COMPLETE = 'complete'

MANUALSKIP = 'manualskip'

PI_ABORT = 'pi_abort'

REASSIGN = 'reassign'

RESUME = 'resume'

SCHEDULE = 'schedule'

START = 'start'

SUSPEND = 'suspend'

UNKNOWN = 'unknown'

WITHDRAW = 'withdraw'

decode(*encoding*)
Decodes any encoding string, referring to the respective standard-model lifecycle transition object in this enum.

Parameters **encoding** (*str*) – Encoding string.
Returns Standard-model transition string
Return type str

values()
Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

Returns An array containing the constants of this enum type, in the order they are declared
Return type list[str]

assign_model(*log*, *model*)
Assigns a value for the lifecycle model identifier to a given log.

Parameters

- **log** (*XLog*) – Log to be tagged.
- **model** (*str*) – Lifecycle model identifier string to be used.

assign_standard_transition(*event*, *transition*)
Assigns a standard lifecycle transition to the given event..

Parameters

- **event** (*XEvent*) – Event to be tagged.
- **transition** (*str*) – Standard lifecycle transition to be assigned.

assign_transition(*event*, *transition*)
Assigns a lifecycle transition string to the given event.

Parameters

- **event** (*XEvent*) – Event to be tagged.
- **transition** (*str*) – Lifecycle transition string to be assigned.

static extract_model(*log*)
Extracts the lifecycle model identifier from a given log.

Parameters **log** (*XLog*) – Event log.

Returns Lifecycle model identifier string.

Return type str

extract_standard_transition(*event*)
Extracts the standard lifecycle transition object from a given event.

Parameters **event** (*XEvent*) – The given event

Returns The standard lifecycle transition instance of this event. Can be null, if not defined.

Return type str

static extract_transition(*event*)
Extracts the lifecycle transition string from a given event.

Parameters **event** (*XEvent*) – The given event

Returns The lifecycle transition string of this event. Can be null, if not defined.

Return type str

uses_standard_model(*log*)
Checks, whether a given log uses the standard model for lifecycle transitions.

Parameters **log** (*XLog*) – Log to be checked.

Returns Returns true, if the log indeed uses the standard lifecycle model.

Return type bool

opyenxes.extension.std.XMicroExtension module

class opyenxes.extension.std.XMicroExtension.**XMicroExtension**
Bases: *opyenxes.extension.XExtension.XExtension*

The micro event extension defines a nesting level, a nesting parent, and the number of nested children for events within a log.

Uses the singleton metaclass

assign_length(*event*, *length*)
Assigns any event its state number of children, as defined by this extension's length attribute.

Parameters

- **event** (*XAttributable*) – Event to assign number of children to.
- **length** (*int*) – The number to be assigned. Should be a non-negative integer.

assign_level(*event*, *level*)
Assigns any event its level, as defined by this extension's level attribute.

Parameters

- **event** (*XAttributable*) – Event to assign level to.
- **level** (*int*) – The level to be assigned. Should be a positive integer.

assign_parent_id(*event*, *parent_id*)
Assigns any event its parent Id, as defined by this extension's parentId attribute.

Parameters

- **event** (*XAttributable*) – Event to assign parent Id to.
- **parent_id** (*XID*) – The parent Id to be assigned. May not be null.

static extract_length(*event*)
Retrieves the stated number of children of an event, if set by this extension's length attribute. Note that this simply returns the value of the "micro:legnth" attribute, and -1 if not present. This does not count the children, it simply returns the number as found in the event.

Parameters **event** (*XEvent*) – Event to extract stated number of children from.

Returns The requested number for this event, -1 if not set.

Return type int

static extract_level(*event*)
Retrieves the level of an event, if set by this extension's level attribute.

Parameters **event** (*XEvent*) – Event to extract level from.

Returns The requested event level, -1 if not set.

Return type int

static extract_parent_id(*event*)
Retrieves the parent Id of an event, if set by this extension's parentId attribute.

Parameters **event** (*XEvent*) – Event to extract parent Id from.

Returns The requested event parent Id, null if not set.

Return type *XID*

static remove_length(*event*)
Removes the stated number of children from an event.

Parameters **event** (*XAttributable*) – The event to remove the number from.

static remove_level(*event*)
Removes the level from an event.

Parameters **event** (*XAttributable*) – The event to remove th elevel from.

static remove_parent_id(*event*)
Removes the parent Id from an event.

Parameters **event** (*XAttributable*) – The event to remove the parent Id from.

opyenxes.extension.std.XOrganizationalExtension module

class opyenxes.extension.std.XOrganizationalExtension.**XOrganizationalExtension**
Bases: *opyenxes.extension.XExtension.XExtension*

This extension adds the organizational perspective to event logs. It defines for events three attributes, referring to:

- The resource which has executed the event
- The role of this resource
- The group of this resource

Uses the singleton metaclass

assign_group(*event*, *instance*)
Assigns the group attribute value for a given event.

Parameters

- **event** (*XEvent*) – Event to be modified.
- **instance** (*str*) – Group string to be assigned.

assign_resource(*event*, *instance*)
Assigns the resource attribute value for a given event.

Parameters

- **event** (*XEvent*) – Event to be modified.
- **instance** (*str*) – Resource string to be assigned.

assign_role(*event*, *instance*)
Assigns the role attribute value for a given event.

Parameters

- **event** (*XEvent*) – Event to be modified.
- **instance** (*str*) – Role string to be assigned.

static extract_group(*event*)
Extracts the group attribute string from an event.

Parameters **event** (*XEvent*) – Event to extract instance from.

Returns The requested activity instance identifier.

Return type str

static extract_resource(*event*)
Extracts the resource attribute string from an event.

Parameters **event** (*XEvent*) – Event to extract instance from.

Returns Resource string for the given event (may be None if not defined)

Return type str

static extract_role(*event*)
Extracts the role attribute string from an event.

Parameters **event** (*XEvent*) – Event to extract instance from.

Returns Role string for the given event (may be None if not defined)

Return type str

opyenxes.extension.std.XSemanticExtension module

class opyenxes.extension.std.XSemanticExtension.**XSemanticExtension**
Bases: *opyenxes.extension.XExtension.XExtension*

This extension adds semantic attributes to event log objects. These semantic attributes reference concepts, which are represented by event log objects, as unique URIs.

Uses the singleton metaclass

assign_model_reference_uris(*target*, *model_reference*)

Assigns to a log element (i.e., archive, log, trace, event, or attribute) a list of model references.

Parameters

- **target** (*XAttributable*) – Any log element (i.e., archive, log, trace, event, or attribute) to be assigned references to.
- **model_reference** (*list[ParseResult or SplitResult]*) – The list of model references, as a list of URIs, referred to by this element.

assign_model_references(*target*, *model_reference*)

Assigns to a log element (i.e., archive, log, trace, event, or attribute) a list of model references.

Parameters

- **target** (*XAttributable*) – Any log element (i.e., archive, log, trace, event, or attribute) to be assigned references to.
- **model_reference** (*list[str]*) – The list of model references, as a list of strings, referred to by this element.

extract_model_reference_uris(*target*)
Retrieves the list of model reference URIs which describe a log element (archive, log, trace, event, attribute).

Parameters **target** (*XAttributable*) – Any log element (i.e., archive, log, trace, event, or attribute) to be queried.

Returns The list of model references, as a list of URIs, referred to by this element.

Return type list[ParseResult]

static extract_model_references(*target*)
Retrieves the list of model references which describe a log element (archive, log, trace, event, attribute).

Parameters **target** (XAttributable) – Any log element (i.e., archive, log, trace, event, or attribute) to be queried.

Returns The list of model references, as a list of strings, referred to by this element.

Return type list[str]

opyenxes.extension.std.XTimeExtension module

class opyenxes.extension.std.XTimeExtension.**XTimeExtension**
Bases: *opyenxes.extension.XExtension.XExtension*

This extension defines the Time perspective on event logs. It makes it possible to assign to each event a timestamp, describing when the event has occurred.

Uses the singleton metaclass

assign_timestamp(*event*, *date*)
Assigns to a given event its timestamp.

Parameters

- **event** (*XEvent*) – Event to be modified.
- **date** (*datetime or int*) – Timestamp, as a datetime object or as a long of milliseconds in UNIX time..

static extract_timestamp(*event*)
Extracts from a given event the timestamp.

Parameters **event** (*XEvent*) – Event to be queried.

Returns The timestamp of this event, as a datetime object (may be null if not defined).

Return type datetime

Module contents

Submodules

opyenxes.extension.XExtension module

class opyenxes.extension.XExtension.**XExtension**(*name*, *prefix*, *uri*)
Bases: object

This class defines and implements extensions to the basic log meta-model. Extensions have a name, a defined prefix, and a unique URI. They can define additional, typed attributes on the level of the log, trace, and event. Also, extensions may define meta attributes

Parameters

- **name** (*str*) – The name of the extension.
- **prefix** (*str*) – Prefix string of the extension, used for addressing attributes.

- **uri** (*urllib.parse.ParseResult or urllib.parse.SplitResult*) – Unique URI of the extension. This URI should point to the file defining the extension, and must be able to be resolved. Extension files should be accessible over the internet, e.g. stored on web servers.

get_defined_attributes()
Returns the collection of attributes defined by this extension for any log elements (archive-, log-, trace-, event-, and meta-attributes).

Returns The collection of attributes defined by this extension

Return type set

get_event_attributes()
Returns the collection of attributes defined by this extension for event elements.

Returns The collection of attributes for event elements

Return type set

get_log_attributes()
Returns the collection of attributes defined by this extension for log elements.

Returns the collection of attributes for log elements

Return type set

get_meta_attributes()
Return the collection of meta-attributes defined by this extension for attributes.

Returns The collection of meta-attributes for attributes.

Return type set

get_name()
Returns the human-readable name of this extension.

Returns The name of this extension.

Return type str

get_prefix()
Returns a unique prefix associated with this extension. This prefix should be no longer than 5 characters, so as not to unnecessarily blow up storage files.

Returns An unique prefix associated with this extension

Return type str

get_trace_attributes()
Returns the collection of attributes defined by this extension for trace elements.

Returns the collection of attributes for trace elements

Return type set

get_uri()
Returns a unique URI associated with this extension. This URI should point to the file defining the extension, and must be able to be resolved. Extension files should be accessible over the internet, e.g. stored on web servers.

Returns An unique URI associated with this extension

Return type urllib.parse.ParseResult or urllib.parse.SplitResult

opyenxes.extension.XExtensionManager module

class opyenxes.extension.XExtensionManager.**XExtensionManager**
Bases: object

The extension manager is used to access, store, and manage extensions in a system. Extensions can be loaded from their given URI, which should point to the file defining the extension. Also, extensions can be registered locally, which then override any remotely-loaded extensions (which are more generic placeholders). Extension files downloaded from remote sources (which happens when the extension cannot be resolved locally) are cached on the local system, so that the network source of extension files is not put under extensive stress. The extension manager is a singleton, there is no need to instantiate more than one extension manager, which is necessary to avoid states of inconsistency.

Uses the singleton metaclass

static cache_extension(*uri*)
Downloads and caches an extension from its remote definition file. The extension is subsequently placed in the local cache, so that future loading is accelerated.

Parameters **uri** (*urllib.parse.ParseResult or urllib.parse.SplitResult*) – Unique URI of the extension which is to be cached.

get_by_index(*index*)
Retrieves an extension by ints index. If no extension with the given index is found, this method returns null.

Parameters **index** (*int*) – The index of the requested extension.

Returns The requested extension (may be null, if it cannot be found).

Return type *XExtension*

get_by_name(*name*)
Retrieves an extension by its name. If no extension by that name can be found, this method returns null.

Parameters **name** (*str*) – The name of the requested extension.

Returns The requested extension (may be null, if it cannot be found).

Return type *XExtension*

get_by_prefix(*prefix*)
Retrieves an extension by its prefix. If no extension by that prefix can be found, this method returns null.

Parameters **prefix** (*str*) – The prefix of the requested extension.

Returns The requested extension (may be null, if it cannot be found).

Return type *XExtension*

get_by_uri(*uri*)
Retrieves an extension instance by its unique URI. If the extension has not been registered before, it is looked up in the local cache. If it cannot be found in the cache, the manager attempts to download it from its unique URI, and add it to the set of managed extensions.

Parameters **uri** (*urllib.parse.ParseResult or urllib.parse.SplitResult*) – The unique URI of the requested extension.

Returns The requested extension.

Return type *XExtension*

get_index(*extension*)
Resolves the index of an extension, given that this extension has been previously registered with this manager instance. If the given index has not been registered previously, this method returns -1.

Parameters **extension** (*XExtension*) – The extension to look up the index for.

Returns Unique index of the requested extension (positive integer).

Return type int

load_extension_cache()
Loads all extensions stored in the local cache. Cached extensions which exceed the maximum caching age are discarded, and downloaded freshly.

register(*extension*)
Explicitly registers an extension instance with the extension manager.

Parameters **extension** (*XExtension*) – The extension to be registered.

register_standard_extensions()
Registers all defined standard extensions with the extension manager before caching.

opyenxes.extension.XExtensionParser module

class opyenxes.extension.XExtensionParser.**XExtensionParser**
Bases: object

Parser for extension definition files.

Uses the singleton metaclass

class XExtensionHandler
Bases: xml.sax.handler.ContentHandler

SAX handler class for extension definition files.

endElement(*local_name*)
Overrides endElement in class ContentHandler
Parameters **local_name** (*str*) – The name of the element type, just as with the startElement event

get_extension()
Retrieves the parsed extension after parsing.
Returns The parsed extension.
Return type *XExtension*

reset()
Resets the handler to initial state.

startElement(*name*, *attributes*)
Overrides startElement in class ContentHandler
Parameters
- **name** (*str*) – Contains the raw XML 1.0 name of the element type
- **attributes** (*xml.sax.xmlreader.AttributesImpl*) – An instance of the Attributes class containing the attributes of the element

static parse(*file*)
Parses an extension from a definition file.

Parameters **file** (*str*) – The path of the file containing the extension or url string which represents the extension definition file..

Returns The extension object, as defined in the provided file.

Return type *XExtension*

Module contents

opyenxes.factory package

Submodules

opyenxes.factory.XFactory module

class opyenxes.factory.XFactory.**XFactory**
Bases: object

Provide methods for creating all element classes of the XES model type hierarchy

static create_attribute_boolean(*key*, *value*, *extension=None*)
Creates a new XES attribute with boolean value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*bool*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new attribute with boolean value.

Return type *XAttributeBoolean*

static create_attribute_container(*key*, *extension=None*)
Creates a new XES container attribute.

Parameters

- **key** (*str*) – The key of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new container attribute.

Return type *XAttributeContainer*

static create_attribute_continuous(*key*, *value*, *extension=None*)
Creates a new XES attribute with float value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*float*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new attribute with float value.

Return type *XAttributeContinuous*

static create_attribute_discrete(*key*, *value*, *extension=None*)
Creates a new XES attribute with integer value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*int*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new attribute with integer value.

Return type *XAttributeDiscrete*

static create_attribute_id(*key*, *value*, *extension=None*)
Creates a new XES attribute with XID value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*XID*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new attribute with XID value.

Return type *XAttributeID*

static create_attribute_list(*key*, *extension=None*)
Creates a new XES list attribute.

Parameters

- **key** (*str*) – The key of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new list attribute.

Return type *XAttributeList*

static create_attribute_literal(*key*, *value*, *extension=None*)
Creates a new XES attribute with string value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*str*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new attribute with string value.

Return type *XAttributeLiteral*

static create_attribute_map()
Creates a new XES attribute map.

Returns A new attribute map instance.

Return type *XAttributeMap*

static create_attribute_timestamp(*key*, *value*, *extension=None*)
Creates a new XES attribute with datetime value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*datetime.datetime or int*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension).

Returns A new attribute with datetime value.

Return type *XAttributeTimestamp*

static create_event(*attribute=None*, *identity=None*)
Creates a new XES event instance.

Parameters

- **attribute** (*XAttributeMap*) – A *XAttributeMap* with the attribute for the event.
- **identity** (*XID*) – The identity defining the attribute (set to None, if the attribute is not associated to an identity).

Returns A new event instance.

Return type *XEvent*

static create_log(*attribute=None*)
Creates a new XES log instance.

Parameters **attribute** (*XAttributeMap*) – A *XAttributeMap* with the attribute for the log.

Returns A new log instance.

Return type *XLog*

static create_trace(*attribute=None*)
Creates a new XES trace instance.

Parameters **attribute** (*XAttributeMap*) – A *XAttributeMap* with the attribute for the trace.

Returns A new trace instance.

Return type *XTrace*

opyenxes.factory.XFactoryRegistry module

class opyenxes.factory.XFactoryRegistry.**XFactoryRegistry**
Bases: *opyenxes.utils.XRegistry.XRegistry*

XModelFactoryRegistry is the most important integration point for external contributors, aside from the extension infrastructure. This singleton class serves as a system-wide registry for XES factory implementations. It provides a current, i.e. standard, factory implementation, which can be switched by applications. This factory will be used in any internal places, e.g., for creating models from reading XES serializations. Other, e.g. proprietary or domain-specific, implementations of the XES standard (and the OpenXES model hierarchy interface) are suggested to implement the XModelFactory interface, and to register their factory with this registry. This enables to transparently switch the storage implementation of the complete OpenXES system (wherever applicable), and every application making use of this registry to create new models.

Module contents

opyenxes.id package

Submodules

opyenxes.id.XID module

class opyenxes.id.XID.**XID**(*uuid_arg=None*)
Bases: object

Implements a unique ID based on UUID.

Parameters **uuid_arg** (*uuid.UUID*) – The UUID implementing XID uniqueness.

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XID*

compare_to(*other*)
Helper method to compares this object with the specified object for order.

Parameters **other** (*XID*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_uuid()
Retrieves the uuid value of this object.

Returns The uuid attribute.

Return type uuid.UUID

static parse(*id_string*)
Parses an XID object from its text representation.

Parameters **id_string** (*str*) – Text representation of an XID.

Returns The parsed XID.

Return type *XID*

set_uuid(*uuid_arg*)
Assigns the uuid value of this object.

Parameters **uuid_arg** (*uuid.UUID*) – The new uuid value.

opyenxes.id.XIDFactory module

class opyenxes.id.XIDFactory.**XIDFactory**
Bases: object

This class is a factory for unique identifiers, as they are used throughout the XES model for element identification.

Uses the singleton metaclass

static create_id()
Creates a new, unique ID

Returns Unique ID

Return type *XID*

Module contents

opyenxes.info package

Submodules

opyenxes.info.XAttributeInfo module

class opyenxes.info.XAttributeInfo.**XAttributeInfo**
Bases: object

This class provides aggregate information about attributes within one container in the log type hierarchy. For example, it may store information about all event attributes in a log.

get_attribute_keys()
Provides access to prototypes of all registered attributes' keys.

Returns A tuple of attribute keys.

Return type tuple

get_attributes()
Provides access to prototypes of all registered attributes.

Returns A tuple of attribute prototypes.

Return type tuple

get_attributes_for_extension(*extension*)
For a given extension, returns prototypes of all registered attributes defined by that extension.

Parameters **extension** (*XExtension*) – Requested attribute extension.

Returns A tuple of attribute prototypes registered for that extension.

Return type tuple

get_attributes_for_type(*type_argument*)
For a given type, returns prototypes of all registered attributes with that type.

Parameters **type_argument** (*type*) – Requested attribute type.

Returns A tuple of attribute prototypes registered for that type.

Return type tuple

get_attributes_without_extension()
Returns prototypes of all registered attributes defined by no extension.

Returns A tuple of attribute prototypes registered for no extension.

Return type tuple

get_frequency(*element*)
Returns the total frequency, i.e. number of occurrences, for the requested attribute.

Parameters **element** (str or *XAttribute*) – Key of an attribute or an attribute.

Returns Total frequency of that attribute as registered.

Return type int

get_keys_for_extension(*extension*)
For a given extension, returns the keys of all registered attributes defined by that extension.

Parameters **extension** (*XExtension*) – Requested attribute extension.

Returns A tuple of attribute keys registered for that extension.

Return type tuple

get_keys_for_type(*type_argument*)
For a given type, returns the keys of all registered attributes with that type.

Parameters **type_argument** (*type*) – Requested attribute type.

Returns A tuple of attribute keys registered for that type.

Return type tuple

get_keys_without_extension()
Returns keys of all registered attributes defined by no extension.

Returns A tuple of attribute keys registered for no extension.

Return type tuple

get_relative_frequency(*element*)
Returns the relative frequency, i.e. between 0 and 1, for the requested attribute.

Parameters **element** (str or *XAttribute*) – Key of an attribute or an attribute.

Returns Relative frequency of that attribute as registered.

Return type int

register(*attribute*)
Registers a concrete attribute with this registry.

Parameters **attribute** (*XAttribute*) – Attribute to be registered.

opyenxes.info.XAttributeNameMap module

class opyenxes.info.XAttributeNameMap.**XAttributeNameMap**(*name*)
Bases: object

Implements an attribute name mapping.

Parameters **name** (*str*) – Name of the mapping.

get_mapping_name()
Returns the name of this mapping.

Returns The name of this mapping.

Return type str

map(*attribute*)
Returns the name mapped onto the provided attribute by this mapping. If no mapping for the given attribute is provided by this map, null is returned.

Parameters **attribute** (*XAttribute* or Str) – Attribute or Attribute key to retrieve mapping for.

Returns The mapping for the given attribute key, or null, if no such mapping exists.

Return type str

register_mapping(*attribute*, *alias*)
Registers a mapping for a given attribute or attribute key.

Parameters

- **attribute** (*XAttribute* or str) – Attribute or attribute key for which to register a mapping.
- **alias** (*str*) – Alias string to map the attribute to.

opyenxes.info.XGlobalAttributeNameMap module

class opyenxes.info.XGlobalAttributeNameMap.**XGlobalAttributeNameMap**
Bases: object

This singleton class implements a global attribute name mapping facility and can manage a number of attribute name mappings. Further, this class also acts as a proxy to the standard mapping, i.e. it can be used directly as a attribute name mapping instance.

MAPPING_DUTCH = 'NL'

MAPPING_ENGLISH = 'EN'

MAPPING_FRENCH = 'FR'

MAPPING_GERMAN = 'DE'

MAPPING_ITALIAN = 'IT'

MAPPING_PORTUGUESE = 'PT'

MAPPING_SPANISH = 'ES'

MAPPING_STANDARD = 'EN'

get_available_mapping_names()
Returns the names of all available mappings. Note that referenced mappings may be empty.

Returns A tuple of names of all available mappings.

Return type tuple

get_available_mappings()
Returns all available mappings. Note that returned mappings may be empty.

Returns A tuple of all available mappings.

Return type tuple

get_mapping(*name*)
Provides access to a specific attribute name mapping by its name. If the requested mapping does not exist yet, a new mapping will be created, added to the set of managed mappings, and returned. This means, this method will always return a mapping, but this could be empty.

Parameters **name** (*str*) – Name of the requested mapping.

Returns The requested mapping, as stored in this facility (or newly created).

Return type *XAttributeNameMap*

static get_mapping_name()
Returns the name of this mapping.

Returns The name of this mapping.

Return type str

get_standard_mapping()
Retrieves the standard attribute name mapping, i.e. the EN english language mapping.

Returns The standard mapping.

Return type *XAttributeNameMap*

map(*attribute*)
Returns the name mapped onto the provided attribute by this mapping. If no mapping for the given attribute is provided by this map, null is returned.

Parameters **attribute** (str or *XAttribute*) – Attribute or attribute key to retrieve mapping for.

Returns The mapping for the given attribute, or null, if no such mapping exists.

Return type str

map_safely(*attribute*, *mapping*)
Maps an attribute safely, using the given attribute mapping. Safe mapping attempts to map the attribute using the given mapping first. If this does not succeed, the standard mapping (EN) will be used for mapping. If no mapping is available in the standard mapping, the original attribute key is returned unchanged. This way, it is always ensured that this method returns a valid string for naming attributes.

Parameters

- **attribute** (*XAttribute* or str) – Attribute to map or key of the attribute to map.
- **mapping** (str or *XAttributeNameMap*) – Name of the mapping to be used preferably or attribute name map to be used preferably.

Returns The safe mapping for the given attribute.

Return type str

register_mapping(*mapping_name*, *attribute_key*, *alias*)
Registers a known attribute for mapping in a given attribute name map. **IMPORTANT**: This method should only be called when one intends to create, or add to, the global attribute name mapping.

Parameters

- **mapping_name** (*str*) – Name of the mapping to register with.
- **attribute_key** (*str*) – Attribute key to be mapped.
- **alias** (*str*) – Alias to map the given attribute to.

opyenxes.info.XLogInfo module

class opyenxes.info.XLogInfo.**XLogInfo**(*log*, *default_classifier*, *classifiers*)
Bases: object

This class implements a bare-bones log info summary which can be created on demand by using applications. The log info summary is based on an event classifier, which is used to identify event class abstractions.

Parameters

- **log** (*XLog*) – The event log to create an info summary for.
- **default_classifier** (*XEventAttributeClassifier*) – The default event classifier to be used
- **classifiers** (list[*XEventAttributeClassifier*]) – A collection of additional event classifiers to be covered by the created log info instance.

LIFECYCLE_TRANSITION_CLASSIFIER = <opyenxes.classification.XEventLifeTransClassifier.X

NAME_CLASSIFIER = <opyenxes.classification.XEventNameClassifier.XEventNameClassifier ob

RESOURCE_CLASSIFIER = <opyenxes.classification.XEventResourceClassifier.XEventResource

STANDARD_CLASSIFIER = <opyenxes.classification.XEventAttributeClassifier.XEventAttribu

static create(*log*, *default_classifier=None*, *classifiers=None*)

Creates a new log info summary with the standard event classifier.

Parameters

- **log** (*XLog*) – The event log to create an info summary for.
- **default_classifier** (*XEventAttributeClassifier*) – The default event classifier to be used
- **classifiers** (list[*XEventAttributeClassifier*]) – A collection of additional event classifiers to be covered by the created log info instance.

Returns The log info summary for this log.

Return type *XLogInfo*

get_event_attribute_info()

Retrieves attribute information about all attributes this log contains on the event level.

Returns Attribute information on the event level.

Return type *XAttributeInfo*

get_event_classes(*classifier=None*)

Retrieves the event classes for a given classifier. *Note:* The given event classifier must be covered by this log info, i.e., the log info must have been created with this classifier. Otherwise, this method will return null. You can retrieve the collection of event classifiers covered by this log info instance by calling the method getEventClassifiers().

Parameters **classifier** (*XEventAttributeClassifier* or None) – The classifier for which to retrieve the event classes.

Returns The requested event classes, or null if the given event classifier is not covered by this log info instance.

Return type *XEventClasses*

get_event_classifiers()

Retrieves the set of event classifiers covered by this log info, i.e., for which event classes are registered in this log info instance.

Returns The tuple of event classifiers covered by this log info instance.

Return type tuple

get_log()

Retrieves the log used for this summary.

Returns The event log which this summary describes.

Return type *XLog*

get_log_attribute_info()
Retrieves attribute information about all attributes this log contains on the log level.

Returns Attribute information on the log level.

Return type *XAttributeInfo*

get_log_time_boundaries()
Retrieves the global timestamp boundaries of this log.

Returns Timestamp boundaries for the complete log.

:rtype *XTimeBounds*

get_meta_attribute_info()
Retrieves attribute information about all attributes this log contains on the meta (i.e., attribute) level.

Returns Attribute information on the meta level.

Return type *XAttributeInfo*

get_name_classes()
Retrieves the event name classes of the summarized log.

Returns The event name classes of the summarized log.

Return type *XEventClasses*

get_number_of_event()
Retrieves the total number of events in this log.

Returns Total number of events.

Return type int

get_number_of_traces()
Retrieves the number of traces in this log.

Returns Number of traces available in this log.

Return type int

get_resource_classes()
Retrieves the resource classes of the summarized log.

Returns The resource classes of the summarized log.

Return type *XEventClasses*

get_trace_attribute_info()
Retrieves attribute information about all attributes this log contains on the trace level.

Returns Attribute information on the trace level.

Return type *XAttributeInfo*

get_trace_time_boundaries(*trace*)
Retrieves the timestamp boundaries for a specified trace.

Parameters **trace** – Trace to be queried for.

Returns Timestamp boundaries for the indicated trace.

Return type *XTimeBounds*

get_transition_classes()
Retrieves the lifecycle transition classes of the summarized log.

Returns The lifecycle transition classes of the summarized log.

Return type *XEventClasses*

register_attributes(*attribute_info*, *attributable*)
Registers all attributes of a given attributable, i.e. model type hierarchy element, in the given attribute info registry.

Parameters

- **attribute_info** (*XAttributeInfo*) – Attribute info registry to use for registration.
- **attributable** (*XAttributable*) – Attributable whose attributes to register.

setup()
Creates the internal data structures of this summary on setup from the log.

opyenxes.info.XLogInfoFactory module

class opyenxes.info.XLogInfoFactory.**XLogInfoFactory**
Bases: object

Factory for deriving log info summaries from logs.

static create_log_info(*log*, *classifier=None*)
Creates a new log info summary with a custom or standard event classifier.

Parameters

- **log** (*XLog*) – The event log to create an info summary for.
- **classifier** (*XEventAttributeClassifier*) – The event classifier to be used.

Returns The log info summary for this log.

Return type *XLogInfo*

opyenxes.info.XTimeBounds module

class opyenxes.info.XTimeBounds.**XTimeBounds**
Bases: object

This class implements timestamp boundaries, which can be used to describe the temporal extent of a log, or of a contained trace.

get_end_date()
Returns the latest timestamp of these boundaries (right bound).

Returns The latest timestamp of these boundaries.

Return type datetime

get_start_date()
Returns the earliest timestamp of these boundaries (left bound).

Returns The earliest timestamp of these boundaries.

Return type datetime

is_within(*date*)
Checks, whether the given date is within these boundaries.

Parameters **date** (*datetime*) – Date to be checked.

Returns Whether the specified date is within these boundaries.

Return type bool

register(*element*)
Registers the given timestamp boundaries. These timestamp boundaries will be potentially adjusted to accomodate for inclusion of the given boundaries.

Parameters **element** (*XTimeBounds* or *XEvent* or datetime) – Timestamp boundaries to be registered.

Module contents

opyenxes.log package

Submodules

opyenxes.log.XLogging module

class opyenxes.log.XLogging.**XLogging**
Bases: object

This class provides low-level logging for library components. Used for debugging.

class Importance
Bases: object

Defines the importance of logging messages.

DEBUG = 'DEBUG'

ERROR = 'ERROR'

INFO = 'INFO'

WARNING = 'WARNING'

log(*message*, *importance=None*)
Logs the given message with debug importance.

Parameters

- **message** – The log message.
- **importance** – Importance of the message.

set_listener(*listener*)
Sets a new logging listener.

Parameters **listener** (*XStdOutLoggingListener*) – New logging listener.

opyenxes.log.XStdoutLoggingListener module

class opyenxes.log.XStdoutLoggingListener.**XStdOutLoggingListener**
Bases: object

Default standard output logging listener.

static log(*message*, *importance*)
Receives an internal OpenXES log message and print that in the console.

Parameters

- **message** (*str*) – Text of the log message.
- **importance** (*str*) – Importance of the log message.

Module contents

opyenxes.model package

Submodules

opyenxes.model.XAttributable module

class opyenxes.model.XAttributable.**XAttributable**(*attribute=None*)
Bases: object

This class is implemented by all elements of the log hierarchy, which can be equipped with attributes

Parameters **attribute** (*XAttributeMap*) – A *XAttributeMap* with the attribute for this class.

get_attributes()
Retrieves the attributes set for this element.

Returns A map of attributes.

Return type *XAttributeMap*

get_extensions()
Retrieves the extensions used by this element, i.e. the extensions used by all attributes of this element, and the element itself

Returns A set of extensions

Return type set(*XExtension*)

has_attributes()
Checks for the existence of attributes

Returns True if this element has any attributes; False otherwise.

Return type bool

set_attributes(*attributes*)
Sets the map of attributes for this element.

Parameters **attributes** (*XAttributeMap*) – A map of attributes

opyenxes.model.XAttribute module

class opyenxes.model.XAttribute.**XAttribute**(*key*, *extension=None*)
Bases: *opyenxes.model.XAttributable.XAttributable*

This class defines attributes used for describing meta-information about event log hierarchy elements. Attributes have a name (i.e., a key), which is string-based

Parameters

- **key** (*str*) – The key of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

compare_to(*other*)
Helper method to compares this object with the specified object for order.

Parameters **other** (*XAttribute*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_extension()
Retrieves the extension defining this attribute.

Returns The extension of this attribute. May return null, if there is no extension defining this attribute.

Return type *XExtension* or None

get_key()
Retrieves the key, i.e. unique identifier, of this attribute

Returns The key of this attribute, as a string.

Return type str

opyenxes.model.XAttributeBoolean module

class opyenxes.model.XAttributeBoolean.**XAttributeBoolean**(*key*, *value*, *extension=None*)
Bases: *opyenxes.model.XAttribute.XAttribute*

Attribute with boolean type value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*bool*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XAttributeBoolean*

compare_to(*obj*)
Compares this object with the specified object for order.

Parameters **obj** (*XAttributeBoolean*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_value()
Retrieves the boolean value of this attribute

Returns Value of this attribute

Return type bool

set_value(*value*)
Assigns the boolean value of this attribute.

Parameters **value** (*bool*) – Value of the attribute.

opyenxes.model.XAttributeCollection module

class opyenxes.model.XAttributeCollection.**XAttributeCollection**(*key*, *extension=None*)
Bases: *opyenxes.model.XAttribute.XAttribute*

This Class is implemented by all attribute that contain more attributes, for example list and container.

Parameters

- **key** (*str*) – The key of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

add_to_collection(*attribute*)
Add attribute in the collection of this object.

Parameters **attribute** (*XAttribute*) – The attribute to add in the collection

get_collection()
Retrieves the list with the attribute of this object.

Returns List of attributes

Return type list(*XAttribute*)

opyenxes.model.XAttributeContainer module

class opyenxes.model.XAttributeContainer.**XAttributeContainer**(*key*, *extension=None*)
Bases: *opyenxes.model.XAttributeCollection.XAttributeCollection*

Attribute with child attributes. Theses child attributes are not ordered.

Parameters

- **key** (*str*) – The key of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

opyenxes.model.XAttributeContinuous module

class opyenxes.model.XAttributeContinuous.**XAttributeContinuous**(*key*, *value*, *extension=None*)
Bases: *opyenxes.model.XAttribute.XAttribute*

Attribute with float type value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*float*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XAttributeContinuous*

compare_to(*obj*)
Helper method to compares this object with the specified object for order.

Parameters **obj** (*XAttributeContinuous*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_value()
Retrieves the float value of this attribute

Returns Value of this attribute

Return type float

set_value(*value*)
Assigns the float value of this attribute.

Parameters **value** (*float*) – Value of the attribute.

opyenxes.model.XAttributeDiscrete module

class opyenxes.model.XAttributeDiscrete.**XAttributeDiscrete**(*key*, *value*, *extension=None*)
Bases: *opyenxes.model.XAttribute.XAttribute*

Attribute with integer type value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*int*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XAttributeDiscrete*

compare_to(*obj*)
Helper method to compares this object with the specified object for order.

Parameters **obj** (*XAttributeDiscrete*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_value()
Retrieves the integer value of this attribute

Returns Value of this attribute

Return type int

set_value(*value*)
Assigns the integer value of this attribute.

Parameters **value** (*int*) – Value of the attribute.

opyenxes.model.XAttributeID module

class opyenxes.model.XAttributeID.**XAttributeID**(*key*, *value*, *extension=None*)
Bases: *opyenxes.model.XAttribute.XAttribute*

Attribute with ID type value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*XID*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XAttributeID*

compare_to(*obj*)
Helper method to compares this object with the specified object for order.

Parameters **obj** (*XAttributeID*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_value()
Retrieves the ID value of this attribute

Returns Value of this attribute

Return type *XID*

set_value(*value*)
Assigns the ID value of this attribute.

Parameters **value** (*XID*) – Value of the attribute.

opyenxes.model.XAttributeList module

class opyenxes.model.XAttributeList.**XAttributeList**(*key*, *extension=None*)
Bases: *opyenxes.model.XAttributeCollection.XAttributeCollection*

Attribute with child attributes. Theses child attributes are ordered.

Parameters

- **key** (*str*) – The key of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

opyenxes.model.XAttributeLiteral module

class opyenxes.model.XAttributeLiteral.**XAttributeLiteral**(*key*, *value*, *extension=None*)
Bases: *opyenxes.model.XAttribute.XAttribute*

Attribute with string type value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*str*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XAttributeLiteral*

compare_to(*obj*)
Helper method to compares this object with the specified object for order.

Parameters **obj** (*XAttributeDiscrete*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_value()
Retrieves the string value of this attribute

Returns Value of this attribute

Return type str

set_value(*value*)
Assigns the string value of this attribute.

Parameters **value** (*str*) – Value of the attribute.

opyenxes.model.XAttributeMap module

class opyenxes.model.XAttributeMap.**XAttributeMap**(*dictionary=None*)
Bases: dict

An attribute map is used to hold a set of attributes, indexed by their key strings, for event log hierarchy elements

Parameters **dictionary** (*dict or None*) – Dictionary with attributes for add in this object.

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XAttributeMap*

is_empty()
Return if this map contains elements

Returns Returns True if this map contains element, False otherwise.

Return type bool

opyenxes.model.XAttributeTimestamp module

class opyenxes.model.XAttributeTimestamp.**XAttributeTimestamp**(*key*, *value*, *extension=None*)
Bases: *opyenxes.model.XAttribute.XAttribute*

Attribute with datetime type value.

Parameters

- **key** (*str*) – The key of the attribute.
- **value** (*datetime or int*) – The value of the attribute.
- **extension** (*XExtension* or None) – The extension defining the attribute (set to None, if the attribute is not associated to an extension)

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XAttributeLiteral*

compare_to(*obj*)
Helper method to compares this object with the specified object for order.

Parameters **obj** (*XAttributeTimestamp*) – the Object to be compared.

Returns A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Return type int

get_value()
Retrieves the datetime value of this attribute

Returns Value of this attribute

Return type datetime

get_value_millis()

set_value(*value*)
Assigns the string value or datetime value of this attribute.

Parameters **value** (*datetime*) – Value of the attribute.

set_value_millis(*value*)

opyenxes.model.XElement module

class opyenxes.model.XElement.**XElement**(*attribute={}*)
Bases: *opyenxes.model.XAttributable.XAttributable*

This Class is implemented by all elements of an event log structure. It defines that all elements are attributable

Parameters **attribute** (*XAttributeMap*) – A *XAttributeMap* with the attribute for this class.

opyenxes.model.XEvent module

class opyenxes.model.XEvent.**XEvent**(*attributes=None*, *identity=None*)
Bases: *opyenxes.model.XElement.XElement*

An event is an element of an XES event log structure. Events are sequentially contained in traces. Events refer to something that has happened during the execution of a process, e.g. the execution of an activity.

Parameters

- **attributes** (*XAttributeMap*) – Map of attribute for the event.
- **identity** (*XID*) – The unique id that represent this event.

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XEvent*

get_id()
Retrieves the id value of this event

Returns id of this event

Return type *XID*

set_id(*idem*)
Assigns the id value of this event.

Parameters **idem** (*XID*) – id of the event.

opyenxes.model.XLog module

class opyenxes.model.XLog.**XLog**(*attributes*)
Bases: *opyenxes.model.XElement.XElement*, list

A log is an element of an XES event log structure. Logs are contained in archives. Any log is a list of traces. Logs represent a collection of traces, which are all representing executions of the same kind of process.

Parameters **attributes** (*XAttributeMap*) – Map of attribute for the log.

append(*p_object*)
Add only a trace object in the log.

Parameters **p_object** (*XTrace*) – a Trace object to append for the log

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XLog*

get_classifiers()
This method returns the list of classifiers defined for this log. This list can be used for reading or writing, i.e., it must be supported to add further classifiers to this list.

Returns The list of classifiers defined for this log.

Return type list[*XEventAttributeClassifier*]

get_extensions()
Retrieves the extensions used by this element, i.e. the extensions used by all attributes of this element, and the element itself

Returns A set of extensions

Return type set(*XExtension*)

get_features()

Retrieves the features of the log for example version, encoding, if have nested-attributes, etc

Returns Dictionary with features

Return type dict

get_global_event_attributes()
This method returns a list of attributes which are global for all events, i.e. every event in the log is guaranteed to have these attributes.

Returns List of ubiquitous trace attributes.

Return type List[*XAttribute*]

get_global_trace_attributes()
This method returns a list of attributes which are global for all traces, i.e. every trace in the log is guaranteed to have these attributes.

Returns List of ubiquitous trace attributes.

Return type List[*XAttribute*]

get_info(*classifier*)
Returns the cached info for the given classifier, null if not available.

Parameters **classifier** (*XEventAttributeClassifier*) – The given classifier

Returns The cached info for the given classifier, null if not available

Return type *XLogInfo* or None

set_features(*key*, *values*)
Assigns the key with the value in de features dictionary.

Parameters

- **key** (*str*) – key of the feature.
- **values** (*Any*) – data of that features

set_info(*classifier*, *info*)
Adds the given info for the given classifier to the info cache.

Parameters

- **classifier** (*XEventClassifier*) – The given classifier.
- **info** (*XLogInfo*) – The given info.

opyenxes.model.XTrace module

class opyenxes.model.XTrace.**XTrace**(*attributes*)
Bases: *opyenxes.model.XElement.XElement*, list

A trace is an element of an XES event log structure. Traces are contained in logs. Any trace is a list of events. Traces describe sequences of events, as they have occurred during one execution of a process, in their given order.

Parameters **attributes** (*XAttributeMap*) – Map of attribute for the trace.

append(*p_object*)
Add only a event object in the log.

Parameters **p_object** (*XEvent*) – a Event object to append for the log

clone()
Creates and returns a copy of this object.

Returns A clone of this instance.

Return type *XTrace*

insert_ordered(*event*)

Insert the event in an ordered manner, if timestamp information is available in this trace.

Parameters **event** (*XEvent*) – the event to be inserted.

Returns index of the inserted event.

Return type int

Module contents

opyenxes.utils package

Submodules

opyenxes.utils.CompareUtils module

opyenxes.utils.CompareUtils.**compare_to_boolean**(*bool_1*, *bool_2*)
This function compares two boolean

Parameters

- **bool_1** – The first boolean to compare.

- **bool_2** – The second boolean to compare with the first

Returns The value 0 if if bool_1 represents the same boolean value as the bool_2 a value less than 0 if bool_1 represents true and bool_2 represents false; and a value greater than 0 if bool_1 represents false and bool_2 represents true.

Return type int

opyenxes.utils.CompareUtils.**compare_to_number**(*number_1*, *number_2*)
This function compares two number, integer or float

Parameters

- **number_1** – The first number to compare.
- **number_2** – The second number to compare with the first

Returns The value 0 if the number_2 is equal to number_1; a value less than 0 if number_2 is greater than number_1; and a value greater than 0 if the number_2 is less than number_1.

Return type int

opyenxes.utils.CompareUtils.**compare_to_string**(*string_1*, *string_2*)
This function compares two string

Parameters

- **string_1** – The first string to compare.
- **string_2** – The second string to compare with the first

Returns The value 0 if the string_2 is lexicographically equal to string_1; a value less than 0 if string_2 is lexicographically greater than string_1; and a value greater than 0 if the string_2 is lexicographically less than string_1.

Return type int

opyenxes.utils.SingletonClassGenerator module

class opyenxes.utils.SingletonClassGenerator.**XConceptExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XConceptExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XCostAmountMetaclass**
Bases: type

This Metaclass produce a singleton XCostAmount class

instance = None

class opyenxes.utils.SingletonClassGenerator.**XCostDriverMetaclass**
Bases: type

This Metaclass produce a singleton XCostDriver class

instance = None

class opyenxes.utils.SingletonClassGenerator.**XCostExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XCostExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XCostTypeMetaclass**
Bases: type

This Metaclass produce a singleton XCostType class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XExtensionManagerMetaclass**
Bases: type

This Metaclass produce a singleton XExtensionManager class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XFactoryRegistryMetaclass**
Bases: type

This Metaclass produce a singleton XFactoryRegistry class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XGlobalAttributeNameMapMetaclass**
Bases: type

This Metaclass produce a singleton XGlobalAttributeNameMap class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XIDFactoryMetaclass**
Bases: type

This Metaclass produce a singleton XIDFactory class

instance = None

class opyenxes.utils.SingletonClassGenerator.**XIdentityExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XIdentityExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XLifecycleExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XLifecycleExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XMicroExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XMicroExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XOrganizationalExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XOrganizationalExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XParserRegistryMetaclass**
Bases: type

This Metaclass produce a singleton XOrganizationalExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XSemanticExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XOrganizationalExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XSerializerRegistryMetaclass**
Bases: type

This Metaclass produce a singleton XOrganizationalExtension class.

instance = None

class opyenxes.utils.SingletonClassGenerator.**XTimeExtensionMetaclass**
Bases: type

This Metaclass produce a singleton XMicroExtension class.

instance = None

opyenxes.utils.XAttributeUtils module

class opyenxes.utils.XAttributeUtils.**XAttributeType**
Bases: enum.Enum

An enumeration.

BOOLEAN = 4

CONTAINER = 7

CONTINUOUS = 3

DISCRETE = 1

ID = 5

LIST = 6

LITERAL = 2

TIMESTAMP = 8

class opyenxes.utils.XAttributeUtils.**XAttributeUtils**
Bases: object

Utilities for working with attributes.

static derive_prototype(*instance*)
Derives a prototype for the given attribute. This prototype attribute will be equal in all respects, expect for the value of the attribute. This value will be set to a default value, depending on the specific type of the given attribute.

Parameters **instance** (*XAttribute*) – Attribute to derive prototype from.

Returns The derived prototype attribute.

Return type *XAttribute*

static get_type(*attribute*)
For the given attribute, returns its type, i.e., the most high-level, typed interface this attribute implements.

Parameters **attribute** (*XAttribute*) – Attribute to analyze.

Returns Class of this attribute.

Return type type

static get_type_string(*attribute*)
For the given attribute, derives the standardized string describing the attributes specific type (used, e.g., for serialization).

Parameters **attribute** (*XAttribute*) – Attribute to extract type string from.

Returns String representation of the attribute's specific type.

Return type str

opyenxes.utils.XRegistry module

class opyenxes.utils.XRegistry.**XRegistry**
Bases: object

Template implementation for a generic registry.

current_default()
Retrieves the current default instance.

Returns The current default instance.

Return type Any

get_available()
Retrieves an unmodifiable set of all available instances.

Returns Tuple of all available instances.

Return type tuple

register(*instance*)
Registers a new instance with this registry.

Parameters **instance** (*Any.*) – Instance to be registered.

set_current_default(*instance*)
Sets the current default instance of this registry.

Parameters **instance** (*Any*) – Instance to be the current default of this registry.

opyenxes.utils.XRuntimeUtils module

class opyenxes.utils.XRuntimeUtils.**XRuntimeUtils**
Bases: object

This class provides runtime utilities for library components. Its main purpose is to identify the host OS, and to locate a standard support folder location on each platform.

determine_os()
Determines the current host platform.

Returns Current host platform.

Return type str

get_extension_cache_folder()
Retrieves the directory file of the platform-dependent OpenXES extension definition file folder.

Returns the directory file of the platform-dependent.

Return type str

get_support_folder()
Retrieves the path of the platform-dependent OpenXES support folder.

Returns The path of the platform-dependent OpenXES support folder.

Return type str

is_running_linux()
Checks whether the current platform is Linux.

Returns True if the current platform is Linux. False otherwise.

Return type bool

is_running_mac_os_x()
Checks whether the current platform is Mac OS X.

Returns True if the current platform is Mac OS X. False otherwise.

Return type bool

is_running_unix()
Checks whether the current platform is some flavor of Unix.

Returns True if the current platform is some flavor of Unix. False otherwise.

Return type bool

is_running_windows()
Checks whether the current platform is Windows.

Returns True if the current platform is Windows. False otherwise.

Return type bool

openxes_version = '1.0RC7'

xes_version = '1.0'

opyenxes.utils.XTimer module

class opyenxes.utils.XTimer.**XTimer**
Bases: object

This class implements a simple timer that can be used to quickly profile the speed of operations within library components. The timer simply uses the system time for timing, and thus does not incur significant overhead on runtime.

DAY_MILLIS = 86400000

HOUR_MILLIS = 3600000

MINUTE_MILLIS = 60000

SECOND_MILLIS = 1000

static format_duration(*millis*)
Formats a duration in milliseconds as a pretty-print string.

Parameters **millis** (*int or float*) – Duration in milliseconds.

Returns Given duration as a pretty-print string.

Return type str

get_duration()
Retrieve the runtime of the timer.

Returns Runtime between start (or creation of timer) and stop, in milliseconds.

Return type int

get_duration_string()
Retrieve the runtime of the timer as a pretty-print string.

Returns Runtime between start (or creation of timer) and stop, as a pretty-print string.

Return type str

start()
Starts the timer.

stop()
Stops the timer (takes time).

opyenxes.utils.XTokenHelper module

class opyenxes.utils.XTokenHelper.**XTokenHelper**
Bases: object

static extract_tokens(*token_string*)

static format_token(*token*)

opyenxes.utils.XsDateTimeConversion module

opyenxes.utils.XsDateTimeConversion.**parse_date_time**(*date_time*)
Transform the date in string format to datetime format

Parameters **date_time** (*str*) – The date in string format

Returns The date in datetime format

Return type datetime

Module contents

4.1.2 Submodules

4.1.3 opyenxes.cli module

4.1.4 Module contents

Top-level package for OpyenXes.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/opyenxes/OpyenXes/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

OpyenXes could always use more documentation, whether as part of the official OpyenXes docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/opyenxes/OpyenXes/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *OpyenXes* for local development.

1. Fork the *OpyenXes* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/OpyenXes.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv opyenxes
$ cd opyenxes/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 opyenxes tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/opyenxes/OpyenXes/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_opyenxes
```

CHAPTER 6

Credits

6.1 Development Lead

- Hernan Valdivieso (The guy who wrote everything!)
- Jorge Munoz-Gama (The mastermind)
- Wai Lam Jonathan Lee

6.2 Contributors

- TKasekamp

CHAPTER 7

History

7.1 0.1.1 (2017-11-08)

- First release on PyPI.

7.2 0.1.2 (2017-11-08)

- Fix missing packages for release.

7.3 0.1.3 (2017-11-08)

- Fixing missing packages in setup.

7.4 0.1.4 (2017-11-08)

- Removed opyenxes module.

7.5 0.1.5 (2017-11-08)

- Fixing missing packages in setup.

7.6 0.1.6 (2017-11-09)

- Making extension.std a module

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

O

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S