EDXML SDK Documentation

Release 2.1.4

D.H.J. Takken

August 12, 2016

Contents

This site documents the edxml Python package, which is part of the EDXML Software Developers Kit. This package offers a reference implementation of the EDXML specification as well as classes for generating, validating and processing of EDXML data streams.

EDXML is a versatile data representation which facilitates implementing data integration solutions. It joins data with semantics to allow data sources to specify the exact meaning of the data they produce. This yields a data representation that combines a broad representational scope with a simple data structure.

Example applications that use this package can be obtained from Github while the EDXML Python package itself can be installed using Pip:

```
pip install edxml
```

Contents:

CHAPTER 1

edxml package

1.1 edxml.EDXMLBase module

This module contains generic (base)classes used throughout the SDK.

exception edxml.EDXMLBase.**EDXMLError**(*message*)
Bases: exceptions.Exception

Generic EDXML exception class

exception edxml.EDXMLBase.**EDXMLProcessingInterrupted**
Bases: exceptions.Exception

Exception for signaling that EDXML processing was aborted

class edxml.EDXMLBase.**EDXMLBase**
Base class for most SDK subclasses

Error(*Message*)
Raises *EDXMLError*.

Parameters **Message** (*str*) – Error message

Warning(*Message*)
Prints a warning to sys.stderr.

Parameters **Message** (*str*) – Warning message

GetWarningCount()
Returns the number of warnings generated

GetErrorCount()
Returns the number of errors generated

ValidateDataType(*ObjectType*, *DataType*)
Validate a data type.

Parameters

- **ObjectType** (*str*) – Name of the object type having specified data type
- **DataType** (*str*) – EDXML data type

calls *Error()* when datatype is invalid.

ValidateObject(*Value*, *ObjectTypeName*, *DataType*, *Regexp=None*)
Validate an object value.

The Value argument can be a string, int, bool, Decimal, etc depending on the data type.

Parameters

- **Value** – Object value.
- **ObjectTypeName** (*str*) – Object type.
- **DataType** (*str*) – EDXML data type of object.
- **Regexp** (*str, optional*) – Regular expression for checking Value.

calls *Error()* when value is invalid.

NormalizeObject(*Value*, *DataType*)

Normalize an object value to a unicode string

Prepares an object value for computing sticky hashes, by applying the normalization rules as outlined in the EDXML specification. It takes a string containing an object value as input and returns a normalized unicode string.

Parameters

- **Value** (*str, unicode*) – The input object value
- **DataType** (*str*) – EDXML data type

Returns unicode. The normalized object value

calls *Error()* when value is invalid.

1.2 edxml.EDXMLDefinitions module

EDXMLDefinitions

This module contains the EDXMLDefinitions class, which manages information from EDXML <definitions> sections.

class edxml.EDXMLDefinitions.**EDXMLDefinitions**

Bases: *edxml.EDXMLBase.EDXMLBase*

Class for managing information from EDXML <definitions> sections.

This class is used for managing definitions of event types, object types and sources from EDXML files. It is used for storing parsed definitions, querying definitions, and merging definitions from various EDXML files. It can be used to store <definitions> sections from multiple EDXML streams in succession, which results in the definitions from all streams being merged together. During the merge, the definitions are automatically checked for compatibility with previously stored definitions. The *edxml.EDXMLBase.EDXMLError* exception is raised when problems are detected.

The class also offers methods to generate EDXML <definitions> sections from the stored definitions, or generate (partial) XSD and RelaxNG schemas which can be used for validation of EDXML files.

SourceIdDefined(*SourceId*)

Returns boolean indicating if given Source ID exists.

Parameters **SourceId** (*str*) – EDXML Source Identifier

Returns bool. Source ID exists (True) or not (False)

EventTypeDefined(*EventTypeName*)

Returns boolean indicating if given event type is defined.

Parameters **EventTypeName** (*str*) – Name of event type

Returns bool. Event type exists (True) or not (False)

PropertyDefined(*EventTypeName*, *PropertyName*)
Returns boolean indicating if given property is defined.

Parameters

- **EventTypeName** (*str*) – Event type name
- **PropertyName** (*str*) – Property Name

Returns bool. Property exists (True) or not (False)

ObjectTypeDefined(*ObjectTypeName*)
Returns boolean indicating if given object type is defined.

Parameters **ObjectTypeName** (*str*) – Object type name

Returns bool. Object type exists (True) or not (False)

RelationDefined(*EventTypeName*, *Property1Name*, *Property2Name*)
Returns boolean indicating if given property relation is defined.

Parameters

- **EventTypeName** (*str*) – Event type name
- **Property1Name** (*str*) – Name of event type property
- **Property2Name** (*str*) – Name of event type property

Returns bool. Relation exists (True) or not (False)

GetRelationPredicates()
Returns list of known relation predicates.

Returns list. List of predicates

EventTypeIsUnique(*EventTypeName*)
Returns a boolean indicating if given eventtype is unique or not.

Parameters **EventTypeName** (*str*) – Name of event type

Returns bool. Event type is unique (True) or not (False)

PropertyIsUnique(*EventTypeName*, *PropertyName*)
Returns a boolean indicating if given property is unique or not.

Parameters

- **EventTypeName** (*str*) – Name of event type
- **PropertyName** (*str*) – Name of event property

Returns bool. Property is unique (True) or not (False)

GetUniqueProperties(*EventTypeName*)
Returns a list of names of unique properties

Parameters **EventTypeName** (*str*) – Name of an event type

Returns list. List of unique properties

GetMandatoryObjectProperties(*EventTypeName*)
Returns a list of names of properties which must have an object

Parameters **EventTypeName** (*str*) – Name of an event type

Returns list. List of mandatory properties

GetSingletonObjectProperties(*EventTypeName*)
Returns a list of names of properties which cannot have multiple objects

Parameters **EventTypeName** (*str*) – Name of an event type

Returns list. List of singleton properties

PropertyDefinesEntity(*EventTypeName*, *PropertyName*)
Returns boolean indicating if property of given event type is an entity identifier.

Parameters

- **EventTypeName** (*str*) – Name of event type
- **PropertyName** (*str*) – Name of event property

Returns bool. Is entity identifier (True) or not (False)

PropertyInRelation(*EventTypeName*, *PropertyName*)
Returns a boolean indicating if given property of specified event type is involved in any defined property relation.

Parameters

- **EventTypeName** (*str*) – Name of event type
- **PropertyName** (*str*) – Name of event property

Returns bool. Is part of relation definition (True) or not (False)

GetSourceURLs()
Returns an ordered list of all parsed source URLs. The order as they appeared in the EDXML stream is preserved.

Returns list. List of EDXML source URLs

GetSourceIDs()
Returns a list of all known source ID

Returns list. List of EDXML source IDs

GetSourceId(*Url*)
Returns the ID of event source having specified URL

Parameters **Url** (*str*) – EDXML source URL

Returns str. EDXML source ID

GetEventTypeNames()
Returns a list of all known event type names. The order as they appeared in the EDXML stream is preserved.

Returns list. List of event type names

GetEventTypeAttributes(*EventTypeName*)
Returns a dictionary containing all attributes of requested event type.

Parameters **EventTypeName** (*str*) – Name of an event type

Returns dict. EDXML attributes

GetEventTypeParent(*EventTypeName*)
Returns a dictionary containing all attributes of the parent of requested eventtype. Returns empty dictionary when event type has no defined parent.

Parameters **EventTypeName** (*str*) – Name of an event type

Returns dict. EDXML attributes of parent

GetEventTypeParentMapping(*EventTypeName*)
Returns a dictionary containing all property names of the event type that map to a parent property. The value of each key corresponds to the name of the parent property that the child property maps to. Returns empty dictionary when event type has no defined parent.

Parameters **EventTypeName** (*str*) – Name of an event type

Returns dict. Child / Parent Property mapping

GetEventTypesHavingObjectType(*ObjectTypeName*)
Returns a list of event type names having specified object type.

Parameters **ObjectTypeName** (*str*) – Name of an object type

Returns list. List of event type names

GetEventTypeNamesInClass(*ClassName*)
Returns a list of event type names that belong to specified class.

Parameters **ClassName** (*str*) – Name of an EDXML event type class

Returns list. List of event type names

GetEventTypeNamesInClasses(*ClassNames*)
Returns a list of event type names that belong to specified list of classes.

Parameters **ClassNames** (*iterable*) – Iterable yielding names of EDXML event type classes

Returns list. List of event type names

GetObjectTypeAttributes(*ObjectTypeName*)
Returns a dictionary containing all attributes of specified object type.

Parameters **ObjectTypeName** (*str*) – Name of an object type

Returns dict. Dictionary of EDXML object type attributes

GetEventTypeProperties(*EventTypeName*)
Returns a list of all property names of given event type. The order as they appeared in the EDXML stream is preserved.

Parameters **EventTypeName** (*str*) – Name of an event type

Returns list. List of event type property names

GetEventTypePropertyRelations(*EventTypeName*)
Returns a list of all IDs of property relations in given event type. The order as they appeared in the EDXML stream is preserved.

Parameters **EventTypeName** (*str*) – Name of an event type

Returns list. List of property relation identifiers

GetPropertyRelationAttributes(*EventTypeName*, *RelationId*)
Returns a dictionary containing all attributes of requested relation.

The returned attributes are the attributes of the EDXML <relation> tag that corresponds to the specified relation identifier.

Parameters

- **EventTypeName** (*str*) – Name of an event type
- **RelationId** (*str*) – Identifier of a property relation

Returns dict. Dictionary containing EDXML attributes of relation tag

GetObjectTypeNames()
Returns a list of all known object type names. The order as they appeared in the EDXML stream is preserved.

Returns list. List of object type names

GetSourceURLProperties(*Url*)
Returns dictionary containing attributes of the source specified by given URL.

The returned dictionary contains the attributes of the EDXML <source> tag

Parameters **Url** (*str*) – EDXML source URL

Returns dict. Dictionary containing the attributes of the source tag.

GetSourceIdProperties(*SourceId*)
Returns dictionary containing attributes of the source specified by given Source ID.

Parameters **SourceId** (*str*) – Source identifier

Returns dict. Dictionary containing the attributes of the source tag.

ObjectTypeRequiresUnicode(*ObjectTypeName*)
Returns True when given string object type requires unicode characters, return False otherwise.

Parameters **ObjectTypeName** (*str*) – Name of an object type

Returns bool. Objects require unicode encoding (True) or not (False)

GetPropertyObjectType(*EventTypeName*, *PropertyName*)
Return the name of the object type of specified event property.

Parameters

- **EventTypeName** (*str*) – Name of event type
- **PropertyName** (*str*) – Name of event property

Returns str. Object type name

GetPropertyAttributes(*EventTypeName*, *PropertyName*)
Return dictionary of attributes of specified event property.

Parameters

- **EventTypeName** (*str*) – Name of event type
- **PropertyName** (*str*) – Name of event property

Returns dict. Dictionary containing the EDXML attributes of the <property> tag

GetObjectTypeDataType(*ObjectTypeName*)
Return the data type of given object type.

Parameters **ObjectTypeName** (*str*) – Name of an object type

Returns str. EDXML data type

AddEventType(*EventTypeName*, *Attributes*)
Add an event type to the collection of event type definitions. If an event type definition with the same name exists, it will be checked for consistency with the existing definition.

Parameters

- **EventTypeName** (*str*) – Name of event type

- **Attributes** (*dict*) – Dictionary holding the attributes of the <eventtype> tag

SetEventTypeParent(*EventTypeName*, *Attributes*)
Configure a parent of specified event type.

Parameters

- **EventTypeName** (*str*) – Name of event type
- **Attributes** (*dict*) – Dictionary holding the attributes of the <parent> tag.

AddProperty(*EventTypeName*, *PropertyName*, *Attributes*)
Add a property to the collection of property definitions. If a property definition with the same name exists, it will be checked for consistency with the existing definition.

Parameters

- **EventTypeName** (*str*) – Name of an event type
- **PropertyName** (*str*) – Name of a property
- **Attributes** (*dict*) – Dictionary holding the attributes of the <property> tag.

AddRelation(*EventTypeName*, *Property1Name*, *Property2Name*, *Attributes*)
Add a relation to the collection of relation definitions. If a relation definition with the same properties exists, it will be checked for consistency with the existing definition.

Parameters

- **EventTypeName** (*str*) – Name of the event type
- **Property1Name** (*str*) – Name of property 1
- **Property2Name** (*str*) – Name of property 2
- **Attributes** (*dict*) – Dictionary holding the attributes of the <relation> tag.

AddObjectType(*ObjectTypeName*, *Attributes*, *WarnNotUsed=True*)
Add an object type to the collection of object type definitions. If an object type definition with the same name exists, it will be checked for consistency with the existing definition.

Parameters

- **ObjectTypeName** (*str*) – Name of event type
- **Attributes** (*str*) – Dictionary holding the attributes of the <objecttype> tag.
- **WarnNotUsed** (*str, optional*) – Generate a warning if no property uses the object type

AddSource(*SourceUrl*, *Attributes*)
Add a source to the collection of event source definitions. If a source definition with the same URL exists, it will be checked for consistency with the existing definition.

Parameters

- **SourceUrl** (*str*) – URL of event source
- **Attributes** (*str*) – Dictionary holding the attributes of the <source> tag.

RemoveSource(*SourceId*)
Remove a source from the collection of event source definitions.

Parameters **SourceId** (*str*) – EDXML source ID

CheckPropertyObjectTypes()
Checks if all object types that properties refer to are defined. Calls self.Error when a problem is detected.

CheckEventTypePropertyConsistency(*EventTypeName*, *PropertyNames*)
Check if specified list of property names is correct for the specified event type. Calls self.Error when a problem is detected.

Parameters

- **EventTypeName** (*str*) – Name of an event type
- **PropertyName** (*str*) – Name of an event property

CheckEventTypeRelations(*EventTypeName*)
Check if the relation definitions for specified eventtype are correct. Calls self.Error when a problem is detected.

CheckEventTypeParents(*EventTypeName*)
Checks if parent definition of given event type is valid, if there is any parent definition.

Parameters **EventTypeName** (*str*) – Name of an event type

CheckReporterString(*EventTypeName*, *String*, *PropertyNames*, *CheckCompleteness=False*)
Checks if given event type reporter string makes sense. Optionally, it can also check if all given properties are present in the string.

Parameters

- **EventTypeName** (*str*) – Name of an event type
- **String** (*str*) – The reporter string
- **PropertyNames** (*list*) – List of property names belonging to the event type
- **CheckCompleteness** (*bool, optional*) – Check if all properties are present in string

UniqueSourceIDs()
Source IDs are required to be unique only within a single EDXML file. When multiple EDXML files are parsed using the same EDXMLParser instance, it may happen that different sources have the same ID. This method changes the Source IDs of all known sources to be unique.

It returns a mapping that maps old Source ID into new Source ID.

Returns dict. Source identifier mapping

MergeEvents(*EventTypeName*, *EventObjectsA*, *EventObjectsB*)
Merges the objects of an event 'B' with the objects of another event 'A'. The arguments EventObjectsA and EventObjectsB should be dictionaries where the keys are property names and the values lists of object values.

The objects in EventObjectsA are updated using the objects from EventObjectsB. It returns True when EventObjectsA was modified, False otherwise.

Note that this method does NOT merge parent hashes, only the property objects.

Parameters

- **EventTypeName** (*str*) – Name of event type of the events
- **EventObjectsA** (*dict*) – Objects of event A
- **EventObjectsB** (*dict*) – Objects of event B

ComputeStickyHash(*EventTypeName*, *EventObjects*, *EventContent*)
Computes a sticky hash from given event. The EventObjects argument should be a list containing dictionaries representing the objects. The dictionaries should contain the property name stored under the 'property' key and the value stored under the 'value' key.

Parameters

- **EventTypeName** (*str*) – The name of the event type
- **EventObjects** (*list*) – List of event objects
- **EventContent** (*str*) – The content of the event

Note: The supplied object values must be normalized using *edxml.EDXMLBase.EDXMLBase.NormalizeObject()*.

Returns str. A hexadecimal string representation of the hash.

ComputeStickyHashV3(*EventTypeName*, *SourceUrl*, *EventObjects*, *EventContent*)

Computes a sticky hash from given event, using the hashing algorithm from EDXML specification version 3.x. The EventObjects argument should be a list containing dictionaries representing the objects. The dictionaries should contain the property name stored under the 'property' key and the value stored under the 'value' key.

Parameters

- **EventTypeName** (*str*) – The name of the event type
- **EventObjects** (*list*) – List of event objects
- **EventContent** (*str*) – The content of the event

Note: The supplied object values must be normalized using *edxml.EDXMLBase.EDXMLBase.NormalizeObject()*.

Returns str. A hexadecimal string representation of the hash.

GenerateEventTypeXML(*EventTypeName*, *XMLGenerator*)

Generates an EDXML fragment which defines specified eventtype. Can be useful for constructing new EDXML files based on existing event type definitions.

The XMLGenerator argument may be either a SAX XMLGenerator instance or a ElementTree SimpleXMLWriter instance.

Parameters

- **EventTypeName** (*str*) – Name of the event type
- **XMLGenerator** (*XMLGenerator, XMLWriter*) – XMLGenerator / XMLWriter instance

GenerateEventPropertyXML(*EventTypeName*, *PropertyName*, *XMLGenerator*)

Generates an EDXML fragment which defines specified eventtype property. Can be useful for constructing new EDXML files based on existing event type definitions.

The XMLGenerator argument may be either a SAX XMLGenerator instance or a ElementTree SimpleXMLWriter instance.

Parameters

- **EventTypeName** (*str*) – Name of the event type
- **PropertyName** (*str*) – Name of the property

- **XMLGenerator** (*XMLGenerator, XMLWriter*) – XMLGenerator / XMLWriter instance

GeneratePropertyRelationsXML(*EventTypeName*, *XMLGenerator*)

Generates an EDXML fragment which defines all property relation of specified eventtype. Can be useful for constructing new EDXML files based on existing event type definitions.

The XMLGenerator argument may be either a SAX XMLGenerator instance or a ElementTree SimpleXMLWriter instance.

Parameters

- **EventTypeName** (*str*) – Name of the event type
- **XMLGenerator** (*XMLGenerator, XMLWriter*) – XMLGenerator / XMLWriter instance

GeneratePropertyRelationXML(*EventTypeName*, *RelationId*, *XMLGenerator*)

Generates an EDXML fragment which defines specified property relation of specified eventtype. Can be useful for constructing new EDXML files based on existing event type definitions.

The XMLGenerator argument may be either a SAX XMLGenerator instance or a ElementTree SimpleXMLWriter instance.

Parameters

- **EventTypeName** (*str*) – Name of the event type
- **RelationId** (*str*) – Identifier of a property relation
- **XMLGenerator** (*XMLGenerator, XMLWriter*) – XMLGenerator / XMLWriter instance

GenerateObjectTypeXML(*ObjectTypeName*, *XMLGenerator*)

Generates an EDXML fragment which defines specified object type. Can be useful for constructing new EDXML files based on existing object type definitions.

The XMLGenerator argument may be either a SAX XMLGenerator instance or a ElementTree SimpleXMLWriter instance.

Parameters

- **EventTypeName** (*str*) – Name of the event type
- **XMLGenerator** (*XMLGenerator, XMLWriter*) – XMLGenerator / XMLWriter instance

GenerateEventSourceXML(*SourceUrl*, *XMLGenerator*)

Generates an EDXML fragment which defines specified event source. Can be useful for constructing new EDXML files based on existing event source definitions.

The XMLGenerator argument may be either a SAX XMLGenerator instance or a ElementTree SimpleXMLWriter instance.

Parameters

- **SourceUrl** (*str*) – EDXML source URL
- **XMLGenerator** (*XMLGenerator, XMLWriter*) – XMLGenerator / XMLWriter instance

GenerateXMLDefinitions(*XMLGenerator*, *IncludeSources=True*)

Generates a full EDXML <definitions> section, containing all known event types, event types and optionally sources.

The XMLGenerator argument may be either a SAX XMLGenerator instance or a ElementTree SimpleXMLWriter instance.

Parameters

- **XMLGenerator** (*XMLGenerator, XMLWriter*) – XMLGenerator / XMLWriter instance
- **IncludeSources** (*bool, optional*) – Boolean, include source definitions yes or no

OpenXSD()
Start generating an XSD schema from stored definitions. Always call this before constructing a (partial) XSD schema.

CloseXSD()
Finalize generated XSD and return it as a string.

GenerateEventTypeXSD(*EventTypeName*)
Generates an XSD fragment related to the event type definition of specified event type. Can be useful for generating modular XSD schemas or constructing full EDXML validation schemas.

Make sure to call *OpenXSD()* first.

Parameters **EventTypeName** (*str*) – Name of an event type

GenerateObjectTypeXSD(*ObjectTypeName*)
Generates an XSD fragment related to the object type definition of specified object type. Can be useful for generating modular XSD schemas or constructing full EDXML validation schemas.

Make sure to call *OpenXSD()* first.

Parameters **ObjectTypeName** (*str*) – Name of an object type

GenerateFullXSD()
Generates an full XSD schema for EDXML files that contain all known definitions of event types, object types and sources.

Make sure to call *OpenXSD()* first.

OpenRelaxNG()
Start generating a RelaxNG schema from stored definitions. Always call this before constructing a (partial) RelaxNG schema.

CloseRelaxNG()
Finalize RelaxNG schema and return it as a string.

Returns str. The RelaxNG schema

GenerateEventTypeRelaxNG(*EventTypeName*)
Generates a RelaxNG fragment related to the event type definition of specified event type. Can be useful for generating modular RelaxNG schemas or constructing full EDXML validation schemas.

Parameters **EventTypeName** (*str*) – Name of an event type

Make sure to call *OpenRelaxNG()* first.

GenerateObjectTypeRelaxNG(*ObjectTypeName*)
Generates a RelaxNG fragment related to the object type definition of specified object type. Can be useful for generating modular RelaxNG schemas or constructing full EDXML validation schemas.

Make sure to call *OpenRelaxNG()* first.

Parameters **ObjectTypeName** (*str*) – Name of an object type

GenerateEventRelaxNG(*EventTypeName*)
Generates a RelaxNG fragment related to the object type definition of specified object type. Can be useful for generating modular RelaxNG schemas or constructing full EDXML validation schemas.

Make sure to call *OpenRelaxNG()* first.

Parameters **EventTypeName** (*str*) – Name of an event type

GenerateGenericSourcesRelaxNG()
Generates a RelaxNG fragment representing an event source. Can be useful for generating modular RelaxNG schemas or constructing full EDXML validation schemas.

Make sure to call *OpenRelaxNG()* first.

GenerateFullRelaxNG(*EventRefs=None*, *EventTypeRefs=None*, *ObjectTypeRefs=None*)
Generates a full RelaxNG schema, containing all known definitions of event types, object types and sources. You can optionally provide dictionaries which map event type names or object type names to URIs. In this case, the resulting schema will refer to these URIs in stead of generating the schema patterns in place. This might be useful if you have a central storage for event type definitions or object type definitions.

Make sure to call *OpenRelaxNG()* first.

Parameters

- **EventRefs** (*dict, optional*) – Dictionary containing URI of event schema for every event type name
- **EventTypeRefs** (*dict, optional*) – Dictionary containing URI of event type schema for every event type name
- **ObjectTypeRefs** (*dict, optional*) – Dictionary containing URI of object type schema for every object type name

1.3 edxml.EDXMLFilter module

EDXMLFilter

This module can be used to write EDXML filtering scripts, which can edit EDXML streams. All filtering classes are based on *edxml.EDXMLParser*, so you can conveniently use Definitions attribute of EDXMLParser to query details about all defined event types, object types, sources, and so on.

class edxml.EDXMLFilter.**EDXMLStreamFilter**(*upstream*, *SkipEvents=False*, *Output=<open file '<stdout>', mode 'w'>*)
Bases: *edxml.EDXMLParser.EDXMLParser*

Base class for implementing EDXML filters

This class inherits from EDXMLParser and causes the EDXML data to be passed through to STDOUT.

You can pass any file-like object using the Output parameter, which will be used to send the filtered data stream to. It defaults to sys.stdout (standard output).

Parameters

- **upstream** – XML source (SaxParser instance in most cases)
- **SkipEvents** (*bool, optional*) – Set to True to parse only the definitions section
- **Output** (*bool, optional*) – An optional file-like object, defaults to sys.stdout

SetOutputEnabled(*YesOrNo*)
This method implements a global switch to turn XML pass through on or off. You can use it to allow certain parts of EDXML files to pass through to STDOUT while other parts are filtered out.

Parameters **YesOrNo** (*bool*) – Output enabled (True) or disabled (False)

class edxml.EDXMLFilter.**EDXMLValidatingStreamFilter**(*upstream*, *SkipEvents=False*, *Output=<open file '<stdout>', mode 'w'>*)
Bases: *edxml.EDXMLParser.EDXMLValidatingParser*

Base class for implementing EDXML filters

This class is identical to the EDXMLStreamFilter class, except that it fully validates each event that is output by the filter.

You can pass any file-like object using the Output parameter, which will be used to send the filtered data stream to. It defaults to sys.stdout (standard output).

Parameters

- **upstream** – XML source (SaxParser instance in most cases)
- **SkipEvents** (*bool, optional*) – Set to True to parse only the definitions section
- **Output** (*bool, optional*) – An optional file-like object, defaults to sys.stdout

SetOutputEnabled(*YesOrNo*)
This method implements a global switch to turn XML pass through on or off. You can use it to allow certain parts of EDXML files to pass through to STDOUT while other parts are filtered out.

Note that the output of the filter is validated, so be careful not to break the EDXML data while filtering it.

Parameters **YesOrNo** (*bool*) – Output enabled (True) or disabled (False)

class edxml.EDXMLFilter.**EDXMLObjectEditor**(*upstream*, *Output=<open file '<stdout>', mode 'w'>*)
Bases: *edxml.EDXMLFilter.EDXMLValidatingStreamFilter*

This class implements an EDXML filter which can be used to edit objects in an EDXML stream. It offers the ProcessObject() method which can be overridden to implement your own object editing EDXML processor.

You can pass any file-like object using the Output parameter, which will be used to send the filtered data stream to. It defaults to sys.stdout (standard output).

Parameters

- **upstream** – XML source (SaxParser instance in most cases)
- **Output** (*optional*) – A file-like object, defaults to sys.stdout

InsertObject(*PropertyName*, *Value*)
Insert a new object into the EDXML stream

This method can be called from implementations of *EditObject()* to add objects to the current event.

Parameters

- **PropertyName** (*str*) – Property of the new object
- **Value** (*str*) – Value of the new object

EditObject(*SourceId*, *EventTypeName*, *ObjectTypeName*, *attrs*)
This method can be overridden to process single objects.

Implementations should return the new object attributes by means of an xml.sax.xmlreader.AttributesImpl object.

Parameters

- **SourceId** (*str*) – EDXML Source Identifier
- **EventTypeName** (*str*) – Name of the event type of current event
- **ObjectTypeName** (*str*) – Object type of the object
- **attrs** (*AttributesImpl*) – XML attributes of the <object> tag

Returns AttributesImpl. Updated XML attributes of the <object> tag

class edxml.EDXMLFilter.**EDXMLEventEditor**(*upstream*, *Output=<open file '<stdout>'*, *mode 'w'>*)

Bases: *edxml.EDXMLFilter.EDXMLValidatingStreamFilter*

This class implements an EDXML filter which can use to edit events in an EDXML stream. It offers the ProcessEvent() method which can be overridden to implement your own event editing EDXML processor.

You can pass any file-like object using the Output parameter, which will be used to send the filtered data stream to. It defaults to sys.stdout (standard output).

Parameters

- **upstream** – XML source (SaxParser instance in most cases)
- **Output** (*optional*) – A file-like object, defaults to sys.stdout

DeleteEvent()

Delete an event while editing

Call this method from *EditEvent()* to delete the event in stead of just editing it.

EditEvent(*SourceId*, *EventTypeName*, *EventObjects*, *EventContent*, *EventAttributes*)

Modifies an event

This method can be overridden to process single events.

The EventObjects parameter is a list of dictionaries. Each dictionary represents one object, containing a 'property' key and a 'value' key.

Parameters

- **SourceId** (*str*) – EDXML source identifier
- **EventTypeName** (*str*) – Name of the event type
- **EventObjects** (*list*) – List of event objects
- **EventContent** (*str*) – Event content string
- **EventAttributes** (*AttributesImpl*) – Sax AttributesImpl object containing <event> tag attributes

Returns tuple. Modified copies of the EventObjects, EventContent and EventAttributes parameters, in that order.

1.4 edxml.EDXMLParser module

EDXMLParser

This module is used for parsing out information about eventtype, objecttype and source definitions from EDXML streams.

The classes contain a Definitions property which is in instance of the EDXMLDefinitions class. All parsed information from the EDXML header is stored there, and you can use it to query information about event types, object types, and so on.

Classes in this module:

EDXMLParser EDXMLValidatingParser

class edxml.EDXMLParser.**EDXMLParser**(*upstream*, *SkipEvents=False*)
Bases: *edxml.EDXMLBase.EDXMLBase*, xml.sax.saxutils.XMLFilterBase

The EDXMLParser class can be used as a content handler for Sax, and has several methods that can be overridden to implement custom EDXML processing scripts. It can optionally skip reading the event data itself if you are only interested in obtaining the definitions. In that case, it will abort XML processing by raising the *edxml.EDXMLBase.EDXMLProcessingInterrupted* exception, which you can catch and handle.

Parameters

- **upstream** – XML source (SaxParser instance in most cases)
- **SkipEvents** (*bool, optional*) – Set to True to parse only the definitions section

Definitions
EDXMLDefinitions – *edxml.EDXMLDefinitions.EDXMLDefinitions* instance

DefinitionsXMLGenerator = None
EDXMLDefinitions instance

EndOfStream()
This method can be overridden to finish processing the event stream.

The parser will call this method when the end of the EDXML stream has been reached.

ProcessEvent(*EventTypeName*, *SourceId*, *EventObjects*, *EventContent*, *Parents*)
This method can be overridden to process events. The EventObjects parameter contains a list of dictionaries, one for each object. Each dictionary has two keys. The 'property' key contains the name of the property. The 'value' key contains the value.

Parameters

- **EventTypeName** (*str*) – The name of the event type
- **SourceId** (*str*) – Event source identifier
- **EventObjects** (*list*) – List of objects
- **EventContent** (*str*) – String containing event content
- **Parents** (*list*) – List of hashes of explicit parent events, as hexadecimal strings

ProcessObject(*EventTypeName*, *ObjectProperty*, *ObjectValue*)
This method can be overridden to process objects.

The method will be called by the parser after reading an object element.

Parameters

- **EventTypeName** (*str*) – The name of the event type
- **ObjectProperty** (*str*) – The name of the object property
- **ObjectValue** (*str*) – String containing object value

DefinitionsLoaded()
This method can be overridden to perform some action as soon as the definitions are read and parsed.

The parser will call it as soon as the <definitions> element has been fully read and parsed. From that moment on, all event type and object type definitions can be access through the Definitions attribute of the parser instance.

GetEventCount(*EventTypeName=None*)
Returns the number of events parsed.

When an event type is passed, only the number of events of this type is returned.

Parameters **EventTypeName** (*str, optional*) – Name of an event type

Returns int. The number of events parsed.

GetWarningCount()
Returns the number of warnings issued

Returns int. The number of warnings issued.

GetErrorCount()
Returns the number of errors issued

Returns int. The number of errors issued.

GetDefinitionsElementAsString()
Returns string representation of the <definitions> element

Should not be called until the definitions tag has been fully fed to the parser.

Returns str. The XML string

class edxml.EDXMLParser.**EDXMLValidatingParser**(*upstream*, *SkipEvents=False*, *ValidateObjects=True*)
Bases: *edxml.EDXMLParser.EDXMLParser*

This class extends the functionality of *EDXMLParser* with thorough checking of the EDXML data. You can use the EDXMLValidatingParser class to parse EDXML data that you don't trust. The class will call *edxml.EDXMLBase.EDXMLError()* when it finds problems in the data. Validation is implemented by overriding *EDXMLParser.DefinitionsLoaded()*, *EDXMLParser.ProcessObject()* and *EDXMLParser.ProcessEvent()*.

Like *edxml.EDXMLParser.EDXMLParser*, it can optionally skip reading the event data itself if you are only interested in obtaining and validating the definitions. In that case, it will abort XML processing by raising the *edxml.EDXMLBase.EDXMLProcessingInterrupted* exception, which you can catch and handle.

Parameters

- **upstream** – XML source (SaxParser instance in most cases)
- **SkipEvents** (*bool, optional*) – Set to True to parse only the definitions section
- **ValidateObjects** (*bool, optional*) – Set to False to skip automatic object value validation

Definitions
EDXMLDefinitions – *edxml.EDXMLDefinitions.EDXMLDefinitions* instance

1.5 edxml.EDXMLWriter module

EDXMLWriter

This module contains the EDXMLWriter class, which is used to generate EDXML streams.

class edxml.EDXMLWriter.**EDXMLWriter**(*Output*, *Validate=True*, *ValidateObjects=True*)

Bases: *edxml.EDXMLBase.EDXMLBase*

Class for generating EDXML streams

The Output parameter is a file-like object that will be used to send the XML data to. This file-like object can be pretty much anything, as long as it has a write() method.

The optional Validate parameter controls if the generated EDXML stream should be autovalidated or not. Automatic validation is enabled by default. This parameter applies to all aspects of EDXML validation, except for object value validation, which is covered by the ValidateObjects parameter.

Enabling object value validation always results in full EDXML validation, regardless of the value of the Validate parameter.

Parameters

- **Output** (*file*) – File-like output object
- **Validate** (*bool, optional*) – Enable output validation (True) or not (False)
- **ValidateObjects** (*bool, optional*) – Enable object validation (True) or not (False)

AddXmlDefinitionsElement(*XmlString*)

Apart from programmatically adding to an EDXML stream, it is also possible to insert plain XML into the stream. Both methods result in full automatic validation of the EDXML stream.

Use this method to insert a full <definitions> element.

Parameters **XmlString** (*str*) – String containing the <definitions> element

AddXmlEventTypeElement(*XmlString*)

Apart from programmatically adding to an EDXML stream, it is also possible to insert plain XML into the stream. Both methods result in full automatic validation of the EDXML stream.

Use this method to insert a full <eventtype> element.

Parameters **XmlString** (*str*) – String containing the <eventtype> element

AddXmlObjectTypeTag(*XmlString*)

Apart from programmatically adding to an EDXML stream, it is also possible to insert plain XML into the stream. Both methods result in full automatic validation of the EDXML stream.

Use this method to insert an <objecttype> tag.

Parameters **XmlString** (*str*) – String containing the <objecttype> tag

AddXmlPropertyTag(*XmlString*)

Apart from programmatically adding to an EDXML stream, it is also possible to insert plain XML into the stream. Both methods result in full automatic validation of the EDXML stream.

Use this method to insert a <property> tag.

Parameters **XmlString** (*str*) – String containing the <property> tag

AddXmlRelationTag(*XmlString*)

Apart from programmatically adding to an EDXML stream, it is also possible to insert plain XML into the stream. Both methods result in full automatic validation of the EDXML stream.

Use this method to insert a <relation> tag.

Parameters **XmlString** (*str*) – String containing the <relation> tag

AddXmlSourceTag(*XmlString*)
Apart from programmatically adding to an EDXML stream, it is also possible to insert plain XML into the stream. Both methods result in full automatic validation of the EDXML stream.

Use this method to insert a <source> tag.

Parameters **XmlString** (*str*) – String containing the <source> tag

AddXmlEventTag(*XmlString*)
Apart from programmatically adding to an EDXML stream, it is also possible to insert plain XML into the stream. Both methods result in full automatic validation of the EDXML stream.

Use this method to insert a full <event> element.

Parameters **XmlString** (*str*) – String containing the <event> element

OpenDefinitions()
Opens the <definitions> element

OpenEventDefinitions()
Opens the <eventtypes> element

OpenEventDefinition(*Name*, *Description*, *ClassList*, *ReporterShort*, *ReporterLong*, *DisplayName='/'*)
Opens an event type definition.

Parameters

- **Name** (*str*) – Name of the eventtype
- **Description** (*str*) – Description of the eventtype
- **ClassList** (*str*) – String containing a comma seperated list of class names
- **ReporterShort** (*str*) – Short reporter string. Please refer to the specification for details.
- **LongReporter** (*str*) – Long reporter string. Please refer to the specification for details.
- **DisplayName** (*str, optional*) – EDXML display-name attribute

AddEventTypeParent(*EventTypeName*, *PropertyMapping*, *ParentDescription*, *SiblingsDescription*)
Adds a parent to an event definition.

Parameters

- **EventTypeName** (*str*) – Name of the parent eventtype
- **PropertyMapping** (*str*) – Value of the EDXML propertymap attribute
- **ParentDescription** (*str*) – Value of the EDXML parent-description attribute
- **SiblingsDescription** (*str*) – Value of the EDXML siblings-description attribute

OpenEventDefinitionProperties()
Opens a <properties> element for defining eventtype properties.

AddEventProperty(*Name*, *ObjectTypeName*, *Description*, *DefinesEntity=False*, *EntityConfidence=0*, *Unique=False*, *Merge='drop'*, *Similar=None*)
Adds a property to an event definition.

Parameters

- **Name** (*str*) – Name of the property
- **ObjectTypeName** (*str*) – Name of the object type

- **Description** (*str*) – Description of the property
- **DefinesEntity** (*bool, optional*) – Property is entity identifier or not
- **EntityConfidence** (*float, optional*) – Floating point confidence
- **Unique** (*bool, optional*) – Property is unique or not
- **Merge** (*str, optional*) – Merge strategy (only for unique properties)
- **Similar** (*str, optional*) – EDXML similar attribute value

CloseEventDefinitionProperties()
Closes a previously opened <properties> section

OpenEventDefinitionRelations()
Opens a <relations> section for defining property relations.

AddRelation(*PropertyName1*, *PropertyName2*, *Type*, *Description*, *Confidence*, *Directed=True*)
Adds a property relation to an event definition.

Parameters

- **PropertyName1** (*str*) – Name of first property
- **PropertyName2** (*str*) – Name of second property
- **Type** (*str*) – EDXML Relation type attribute
- **Description** (*str*) – Relation description
- **Confidence** (*float*) – Floating point confidence value
- **Directed** (*bool, optional*) – Boolean indicating if relation is directed (True) or not (False)

CloseEventDefinitionRelations()
Closes a previously opened <relations> section

CloseEventDefinition()
Closes a previously opened event definition

CloseEventDefinitions()
Closes a previously opened <eventtypes> section

OpenObjectTypes()
Opens a <objecttypes> section for defining object types.

AddObjectType(*Name*, *Description*, *ObjectDataType*, *FuzzyMatching='none'*, *DisplayName='/'*, *Compress=False*, *ENP=0*, *Regexp='[\\s\\S]*'*)
Adds a object type definition.

Parameters

- **Name** (*str*) – Name of object type
- **Description** (*str*) – Description of object type
- **ObjectDataType** (*str*) – EDXML Data type
- **FuzzyMatching** (*str, optional*) – EDXML fuzzy-matching attribute
- **DisplayName** (*str*) – Display name
- **Compress** (*bool, optional*) – Use data compression (True) or not (False)
- **ENP** (*int, optional*) – EDXML enp attribute
- **Regexp** (*str, optional*) – EDXML regexp attribute

CloseObjectTypes()
Closes a previously opened <objecttypes> section

OpenSourceDefinitions()
Opens a <sources> section for defining event sources.

AddSource(*SourceId*, *URL*, *DateAcquired*, *Description*)
Adds a source definition.

Parameters

- **SourceId** (*str*) – EDXML source ID
- **URL** (*str*) – Source URL
- **DateAcquired** (*str*) – Acquisition date (yyyymmdd)
- **Description** (*str*) – Description of the source

CloseSourceDefinitions()
Closes a previously opened <sources> section

CloseDefinitions()
Closes the <definitions> section

OpenEventGroups()
Opens the <eventgroups> section, containing all eventgroups

OpenEventGroup(*EventTypeName*, *SourceId*)
Opens an event group.

Parameters

- **EventTypeName** (*str*) – Name of the eventtype
- **SourceId** (*str*) – Source Id

CloseEventGroup()
Closes a previously opened event group

AddEvent(*PropertyObjects*, *Content=''*, *ParentHashes=[]*, *IgnoreInvalidObjects=False*)
Alternative method for adding an event

This method expects a dictionary containing a list of object values for every property.

The optional ParentHashes parameter may contain a list of sticky hashes of explicit parent events, in hexadecimal string representation.

if IgnoreInvalidObjects is set to True, any errors thrown by the validator as a result of invalid object values will be ignored, and the object will not be included in the event.

Parameters

- **PropertyObjects** (*dict*) – Object dictionary
- **Content** (*str,optional*) – Event content
- **ParentHashes** (*list,optional*) – List of explicit parent events
- **IgnoreInvalidObjects** (*bool,optional*) – Option to ignore invalid object values

OpenEvent(*ParentHashes=[]*)
Opens an event.

The optional ParentHashes parameter may contain a list of sticky hashes of explicit parent events, in hexadecimal string representation.

Parameters **ParentHashes** (*list, optional*) – List of explicit parent events

AddObject(*PropertyName*, *Value*, *IgnoreInvalid=False*)
Adds an object to previously opened event.

if IgnoreInvalidObjects is set to True, any errors thrown by the validator as a result of invalid object values will be ignored, and the object will not be included in the event.

Parameters

- **PropertyName** (*str*) – Name of object property
- **Value** – Object value, can be any object that can be converted to unicode.
- **IgnoreInvalid** (*bool, optional*) – Generate a warning in stead of an error for invalid values

AddContent(*ContentString*)
Adds plain text content to previously opened event.

Parameters **ContentString** (*str*) – Event content

AddTranslation(*Language*, *Interpreter*, *TranslationString*)
Adds translated content to previously opened event.

Parameters

- **Language** (*str*) – ISO 639-1 language code
- **Interpreter** (*str*) – Name of interpreter
- **TranslationString** (*str*) – The translation

CloseEvent()
Closes a previously opened event

CloseEventGroups()
Closes a previously opened <eventgroups> section

Indices and tables

- genindex
- modindex
- search

e

A

C

D

E

G

I

M

N

O

P

R

S

U

V

W