pydifact

Apr 22, 2023

Contents:

Pydifact is a library that aims to provide complete support for reading and writing EDIFACT files. These file format, despite being old, is still a standard in many business cases. In Austria e.g., it is used for the transfer of medical reports between medical doctors.

This documentation ATM just contains the API as a starting point.

CHAPTER 1

API

1.1 SegmentCollection

class pydifact.segmentcollection.**AbstractSegmentsContainer**(*extra_header_elements: List[Union[str, List[str]]] = None*)

Represent a collection of EDI Segments for both reading and writing.

You should not instantiate AbstractSegmentsContainer itself, but subclass it use that.

The segments list in AbstractSegmentsContainer includes header and footer segments too. Inheriting envelopes must NOT include these elements in .segments, as get_header_element() and get_footer_element() should provide these elements on-the-fly.

Inheriting classes must set HEADER_TAG and FOOTER_TAG

add_segment(*segment: pydifact.segments.Segment*) → pydifact.segmentcollection.AbstractSegmentsContainer

Append a segment to the collection.

Note: skips segments that are header oder footer tags of this segment container type. :param segment: The segment to add

add_segments(*segments: Union[List[pydifact.segments.Segment], collections.abc.Iterable]*) → pydifact.segmentcollection.AbstractSegmentsContainer

Add multiple segments to the collection. Passing a UNA segment means setting/overriding the control characters and setting the serializer to output the Service String Advice. If you wish to change the control characters from the default and not output the Service String Advice, change self.characters instead, without passing a UNA Segment.

Parameters **segments** (*list or iterable of Segments*) – The segments to add

classmethod from_segments(*segments: Union[List[T], collections.abc.Iterable]*) → pydifact.segmentcollection.AbstractSegmentsContainer

Create a new AbstractSegmentsContainer instance from a iterable list of segments.

Parameters **segments** (*list/iterable of Segment*) – The segments of the EDI interchange

classmethod from_str(*string: str*) → pydifact.segmentcollection.AbstractSegmentsContainer
Create an instance from a string.

This method is intended for usage in inheriting classes, not it AbstractSegmentsContainer itself. :param string: The EDI content

get_footer_segment() → Optional[pydifact.segments.Segment]
Craft and return this container footer segment (if any)

Returns None if there is no footer for that container

get_header_segment() → Optional[pydifact.segments.Segment]
Craft and return this container header segment (if any)

Returns None if there is no header for that container

get_segment(*name: str*, *predicate: collections.abc.Callable = None*) → Optional[pydifact.segments.Segment]
Get the first segment that matches the requested name.

Returns The requested segment, or None if not found

Parameters

- **name** – The name of the segment to return
- **predicate** – Optional predicate that must match on the segments to return

get_segments(*name: str*, *predicate: collections.abc.Callable = None*) → list
Get all the segments that match the requested name.

Parameters

- **name** – The name of the segments to return
- **predicate** – Optional predicate callable that returns True if the given segment matches a condition

Return type list of Segment

serialize(*break_lines: bool = False*) → str
Serialize all the segments added to this object.

Parameters **break_lines** – if True, insert line break after each segment terminator.

split_by(*start_segment_tag: str*) → collections.abc.Iterable
Split a segment collection by tag.

Everything before the first start segment is ignored, so if no matching start segment is found at all, returned result is empty.

Parameters **start_segment_tag** – the segment tag we want to use as separator

Returns generator of segment collections. The start tag is included in each yielded collection

validate()
Validates this container.

This method must be overridden in implementing subclasses, and should make sure that the container is implemented correctly.

It does not return anything and should raise an Exception in case of errors.

class pydifact.segmentcollection.**FileSourcableMixin**
For backward compatibility

For v0.2 drop this class and move from_file() to Interchange class.

classmethod from_file(*file: str, encoding: str = 'iso8859-1'*) → pydifact.segmentcollection.FileSourcableMixin

Create a Interchange instance from a file.

Raises FileNotFoundError if filename is not found. :param encoding: an optional string which specifies the encoding. Default is "iso8859-1". :param file: The full path to a file that contains an EDI message. :rtype: FileSourcableMixin

class pydifact.segmentcollection.**Interchange**(*sender: str, recipient: str, control_reference: str, syntax_identifier: Tuple[str, int], delimiters: pydifact.control.characters.Characters = ':+,? ', timestamp: datetime.datetime = None, *args, **kwargs*)

An interchange (started by UNB segment, ended by UNZ segment)

Optional features of UNB are not yet supported.

Functional groups are not yet supported

Messages are supported, see get_message() and get_message(), but are optional: interchange segments can be accessed without going through messages.

https://www.stylusstudio.com/edifact/40100/**UNB**_.htm https://www.stylusstudio.com/edifact/40100/**UNZ**_.htm

classmethod from_segments(*segments: Union[list, collections.abc.Iterable]*) → pydifact.segmentcollection.Interchange

Create a new AbstractSegmentsContainer instance from a iterable list of segments.

Parameters **segments** (*list/iterable of Segment*) – The segments of the EDI interchange

get_footer_segment() → pydifact.segments.Segment

:returns a (UNZ) footer segment with correct segment count and control reference.

It counts either of the number of messages or, if used, of the number of functional groups in an interchange (TODO).

get_header_segment() → pydifact.segments.Segment

Craft and return this container header segment (if any)

Returns None if there is no header for that container

get_messages() → List[pydifact.segmentcollection.Message]

parses a list of messages out of the internal segments.

:raises EDISyntaxError if constraints are not met (e.g. UNH/UNT both correct)

TODO: parts of this here are better done in the validate() method

validate()

Validates this container.

This method must be overridden in implementing subclasses, and should make sure that the container is implemented correctly.

It does not return anything and should raise an Exception in case of errors.

class pydifact.segmentcollection.**Message**(*reference_number: str, identifier: Tuple, *args, **kwargs*)

A message (started by UNH segment, ended by UNT segment)

Optional features of UNH are not yet supported.

https://www.stylusstudio.com/edifact/40100/UNH_.htm https://www.stylusstudio.com/edifact/40100/UNT_.htm

get_footer_segment() → pydifact.segments.Segment
Craft and return this container footer segment (if any)

Returns None if there is no footer for that container

get_header_segment() → pydifact.segments.Segment
Craft and return this container header segment (if any)

Returns None if there is no header for that container

validate()
Validates the message.

:raises EDISyntaxError in case of syntax errors in the segments

version
Gives version number and release number.

Returns message version, parsable by pkg_resources.parse_version()

class pydifact.segmentcollection.**RawSegmentCollection**(*extra_header_elements: List[Union[str, List[str]]] = None*)
A way to analyze arbitrary bunch of edifact segments.

Similar to the deprecated SegmentCollection, but lacking from_file() and UNA support.

If you are handling an Interchange or a Message, you may want to prefer those classes to RawSegmentCollection, as they offer more features and checks.

validate()
This is just a stub method, no validation done here.

class pydifact.segmentcollection.**SegmentCollection**(**args*, ***kwargs*)
For backward compatibility. Drop it in v0.2

Will be replaced by Interchange or RawSegmentCollection depending on the need.

add_segment(*segment: pydifact.segments.Segment*) → pydifact.segmentcollection.SegmentCollection
Append a segment to the collection. Passing a UNA segment means setting/overriding the control characters and setting the serializer to output the Service String Advice. If you wish to change the control characters from the default and not output the Service String Advice, change self.characters instead, without passing a UNA Segment.

Parameters **segment** – The segment to add

classmethod from_file(**args*, ***kwargs*) → pydifact.segmentcollection.SegmentCollection
Create a Interchange instance from a file.

Raises FileNotFoundError if filename is not found. :param encoding: an optional string which specifies the encoding. Default is "iso8859-1". :param file: The full path to a file that contains an EDI message. :rtype: FileSourcableMixin

class pydifact.segmentcollection.**UNAHandlingMixin**
For backward compatibility

For v0.2 drop this class and move add_segment() to Interchange class.

add_segment(*segment: pydifact.segments.Segment*) → pydifact.segmentcollection.UNAHandlingMixin
Append a segment to the collection. Passing a UNA segment means setting/overriding the control characters and setting the serializer to output the Service String Advice. If you wish to change the control

characters from the default and not output the Service String Advice, change self.characters instead, without passing a UNA Segment.

Parameters **segment** – The segment to add

1.2 Parser

class pydifact.parser.**Parser**(*factory: pydifact.segments.SegmentFactory = None*)

Parse EDI messages into a list of segments.

convert_tokens_to_segments(*tokens: list*, *characters: pydifact.control.characters.Characters*, *with_una: bool = False*)

Convert the tokenized message into an array of segments. :param tokens: The tokens that make up the message :param characters: the control characters to use :param with_una: whether the UNA segment should be included :type tokens: list of Token :rtype list of Segment

static get_control_characters(*message: str*, *characters: pydifact.control.characters.Characters = None*) → Optional[pydifact.control.characters.Characters]

Read the UNA segment from the passed string and extract/store the control characters from it.

Parameters

- **message** – a valid EDI message string, or UNA segment string, to extract the control characters from.
- **characters** – the control characters to use, if none found in the message. Default: ":+,? '"

Returns the control characters

parse(*message: str*, *characters: pydifact.control.characters.Characters = None*) → Generator[pydifact.segments.Segment, Any, None]

Parse the message into a list of segments.

Parameters

- **characters** – the control characters to use, if there is no UNA segment present
- **message** – The EDI message

Return type

1.3 Segments

class pydifact.segments.**Segment**(*tag: str*, **elements*)

Represents a low-level segment of an EDI interchange.

This class is used internally. read-world implementations of specialized should subclass Segment and provide the *tag* and *validate* attributes.

validate() → bool

Segment validation.

The Segment class is part of the lower level interfaces of pydifact. So it assumes that the given parameters are correct, there is no validation done here. However, in segments derived from this class, there should be validation.

Returns bool True if given tag and elements are a valid EDIFACT segment, False if not.

class pydifact.segments.**SegmentFactory**
Factory for producing segments.

static create_segment(*name: str*, **elements*, *validate: bool = True*) → pydifact.segments.Segment
Create a new instance of the relevant class type.

Parameters

- **name** – The name of the segment
- **elements** – The data elements for this segment
- **validate** – bool if True, the created segment is validated before return

class pydifact.segments.**SegmentProvider**
This is a plugin mount point for Segment plugins which represent a certain EDIFACT Segment.

Classes implementing this PluginMount should provide the following attributes:

validate() → bool
Validates the Segment.

1.4 Token

class pydifact.token.**Token**(*token_type: pydifact.token.Token.Type*, *value: str*)
Represents a block of characters in the message.

This could be content, a data separator (usually +), a component data separator (usually :), or a segment terminator (usually ‘).

class Type
An enumeration.

1.5 Serializer

class pydifact.serializer.**Serializer**(*characters: pydifact.control.characters.Characters = None*)
Serialize a bunch of segments into an EDI message string.

escape(*string: Optional[str]*) → str
Escapes control characters.

Parameters **string** – The string to be escaped

serialize(*segments: List[pydifact.segments.Segment]*, *with_una_header: bool = True*, *break_lines=False*) → str
Serialize all the passed segments.

Parameters

- **segments** – A list of segments to serialize
- **with_una_header** – includes/adds an UNA header if set to True (=default) If the segments list contains a UNA header, it is taken, else the default character set is created.
- **break_lines** – if True, insert line break after each segment terminator.

1.6 Tokenizer

class pydifact.tokenizer.**Tokenizer**
Convert EDI messages into tokens for parsing.

end_of_message() → bool
Check if we've reached the end of the message

extract_stored_chars() → str
Return the previously stored characters and empty the store.

get_next_char() → Optional[str]
Get the next character from the message.

get_next_token() → Optional[pydifact.token.Token]
Get the next token from the message.

get_tokens(*message: str*, *characters: pydifact.control.characters.Characters = None*) → List[pydifact.token.Token]
Convert the passed message into tokens. :param characters: the Control Characters to use for tokenizing. If omitted, use a default set. :param message: The EDI message :return: Token[]

is_control_character() → bool
Check if the current character is a control character.

read_next_char() → None
Read the next character from the message.

If the character is an escape character, set the isEscaped flag to True, get the one after it and store that character in the internal storage.

store_current_char_and_read_next() → None
Store the current character and read the next one from the message.

1.7 Plugin API

Pydifact provides a framework, where some classes can be extended via plugins. These basically follow Marty Alchin's Simple Plugin Framework.

The base meta class is a PluginMount:

class pydifact.api.**PluginMount**(*name*, *bases*, *attrs*)
Generic plugin mount point (= entry point) for pydifact plugins.

Note: Plugins that have an **__omitted__** attriute are not added to the list!

Available entry points for plugins are:

class pydifact.segments.**SegmentProvider**
This is a plugin mount point for Segment plugins which represent a certain EDIFACT Segment.

Classes implementing this PluginMount should provide the following attributes:

__str__()
Returns the user readable text representation of this segment.

validate() → bool
Validates the Segment.

Base is the official UNECE reference syntax rules.

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

C

E

F

G

I

M

P

R

S

T

U

V